Target CoDeSys

Reference Manual

update: 20-12-2014

Table of Contents

1. Target libraries 7
1.1 The SysLibCom library (SysLibCom.lib) 8
1.1.1 SysComFlush 9
1.1.2 SysComPurge 10
1.1.3 SysComReadLine 11
1.1.4 SysComRxNum 12
1.2 The SysLibSockets library (SysLibSockets.lib) 13
1.2.1 TYPE INADDR 14
1.2.2 TYPE SOCK_IP_MREQ 15
1.2.3 TYPE SOCKADDRESS 16
1.2.4 TYPE SOCKET_FD_SET 17
1.2.5 TYPE SOCKET_LINGER 18
1.2.6 TYPE SOCKET_TIMEVAL 19
1.2.7 SysSockAccept 20
1.2.8 SysSockBind 21
1.2.9 SysSockClose 22
1.2.10 SysSockConnect 23
1.2.11 SysSockCreate 24
1.2.12 SysSockGetHostByName 25
1.2.13 SysSockGetHostName 26
1.2.14 SysSockGetLastErrorSync 27
1.2.15 SysSockGetOption 29
1.2.16 SysSockHtonl 31
1.2.17 SysSockHtons 32
1.2.18 SysSockInetAddr 33
1.2.19 SysSockInetNtoa 34
1.2.20 SysSockIoctl 35
1.2.21 SysSockListen 37
1.2.22 SysSockNtohl 38
1.2.23 SysSockNtohs 39
1.2.24 SysSockRecv 40
1.2.25 SysSockRecvFrom 41
1.2.26 SysSockSelect 42

1.2.27 SysSockSend 43
1.2.28 SysSockSendTo 44
1.2.29 SysSockSetIPAddress 45
1.2.30 SysSockSetOption 46
1.2.31 SysSockShutdown 47
1.3 The RTOS functions library (rtos.lib) 48
1.3.1 TYPE MSG_EX 49
1.3.2 RtosCreateMsg 50
1.3.3 RtosDeleteMsg 51
1.3.4 RtosDhcpUse 52
1.3.5 RtosFindMsg 53
1.3.6 RtosGetBootstrapVersion 54
1.3.7 RtosGetDevicenames 55
1.3.8 RtosGetDhcpStat 56
1.3.9 RtosGetIniEntry 57
1.3.10 RtosGetIniEntryEx 58
1.3.11 RtosGetLinkstate 59
1.3.12 RtosGetMacAddress 60
1.3.13 RtosGetMsg 61
1.3.14 RtosGetRebootReason 62
1.3.15 RtosGetVersion 63
1.3.16 RtosGetVersionString 64
1.3.17 RtosIpconfig 65
1.3.18 RtosReboot 66
1.3.19 RtosSendMsg 67
1.3.20 RtosServers 68
1.3.21 RtosSetIniEntry 69
1.3.22 RtosSetIniEntryEx 70
1.4 The SysLibEvent library (SysLibEvent.lib) 71
1.4.1 SysEventCreate 72
1.4.2 SysEventDelete 73
1.4.3 SysEventSet 74
1.4.4 SysEventWait 75
1.5 The SysLibCallback library (SysLibCallback.lib) 76
1.6 The SysLibDir library (SysLibDir.lib) 77
1.7 The SysLibPlcCtrl library (SysLibPlcCtrl.lib) 78

1.8 The SysLibStr library (SysLibStr.lib) 79
1.9 The FRAM library (FRAM_Lib.lib) 80
1.9.1 FRAM_Read 81
1.9.2 FRAM_Write 82
1.10 The RTC library (RTC_Lib.lib) 83
1.10.1 DateTime_Read 84
1.10.2 RTC_Read 85
1.10.3 RTC_Write 86
1.11 The MODEM library (MODEM_Lib.lib) 87
1.11.1 DYNDNS_STATUS 88
1.11.2 MODEM_CHAR_SET 89
1.11.3 MODEM_COMMAND 90
1.11.4 MODEM_DATA_SERVICE 91
1.11.5 MODEM_STATUS 92
1.11.6 DynDNS_Client 94
1.11.7 GM01_PPP 96
1.11.8 GM01_PPP_SMS_Call 99
1.11.9 GM01_SMS 103
1.12 The TCPIP library (TCPIP_Lib.lib) 106
1.12.1 HTTP_Get 107
1.12.2 HTTP_Post 109
1.12.3 SendMail 111
1.12.4 SNTP_DateTime 113
1.13 The MySQL library (MySQL_Lib.lib) 114
1.13.1 MySQL_Connect_Set 115
1.13.2 MySQL_Data_Seek 116
1.13.3 MySQL_Database_Set 117
1.13.4 MySQL_Fetch_Row 118
1.13.5 MySQL_Field_Len 119
1.13.6 MySQL_Field_Name 120
1.13.7 MySQL_Num_Fields 121
1.13.8 MySQL_Num_Rows 122
1.13.9 MySQL_Query 123
1.13.10 MySQL_Result 124
1.14 The RS485/Serial library (rs485.lib) 125
1.14.1 TYPE RS485_BREAK 126

1.14.2 TYPE RS485_FLOWCTRL 127
1.14.3 TYPE RS485_MODE 128
1.14.4 TYPE RS485_PARITY 129
1.14.5 TYPE RS485_PORTS 130
1.14.6 Rs485ComClose 131
1.14.7 Rs485ComOpen 132
1.14.8 Rs485FlushOutput 133
1.14.9 Rs485GetStatus 134
1.14.10 Rs485IsByteAvailable 135
1.14.11 Rs485PurgeInput 136
1.14.12 Rs485PurgeOutput 137
1.14.13 Rs485ReceiveBlock 138
1.14.14 Rs485ReceiveByte 139
1.14.15 Rs485SendBlock 140
1.14.16 Rs485SendBreak 141
1.14.17 Rs485SendByte 142
1.14.18 Rs485SetFlowcontrol 143
1.14.19 Rs485SetMode 144
1.15 The RS485/Expansion library (RS485_Lib.lib) 145
1.15.1 DMX512_Tx 146
1.15.2 RS485_Init_A 147
1.15.3 RS485_Tx_Rx_A 148
1.16 The MODBUS library (MODBUS_Lib.lib) 149
1.16.1 MB_RTU_PARITY 150
1.16.2 MODBUS_STATUS 152
1.16.3 MB_RTU_Deinit 153
1.16.4 MB_RTU_Init 154
1.16.5 MB_RTU_Rd_Coils 155
1.16.6 MB_RTU_Rd_Hold_Regs 156
1.16.7 MB_RTU_Rd_Input_Regs 157
1.16.8 MB_RTU_Rd_Inputs 158
1.16.9 MB_RTU_Req_Rsp 159
1.16.10 MB_RTU_Wr_Coils 161
1.16.11 MB_RTU_Wr_Hold_Regs 162
1.16.12 MB_RTU_Wr_Rd_Hold_Regs 163
1.16.13 MB_RTU_Wr_Single_Coil 165

1.16.14 MB_RTU_Wr_Single_Reg 166
1.16.15 MB_RTU_Slave 167
1.16.16 MB_TCP_Connect 170
1.16.17 MB_TCP_Disconnect 172
1.16.18 MB_TCP_Rd_Coils 173
1.16.19 MB_TCP_Rd_Hold_Regs 174
1.16.20 MB_TCP_Rd_Input_Regs 175
1.16.21 MB_TCP_Rd_Inputs 176
1.16.22 MB_TCP_Req_Rsp 177
1.16.23 MB_TCP_Wr_Coils 180
1.16.24 MB_TCP_Wr_Hold_Regs 181
1.16.25 MB_TCP_Wr_Rd_Hold_Regs 182
1.16.26 MB_TCP_Wr_Single_Coil 184
1.16.27 MB_TCP_Wr_Single_Reg 185
1.16.28 MB_TCP_Server 186
1.16.29 MB_Swap_Dword 189
1.16.30 MB_Swap_Dword2 190
1.16.31 MB_Swap_Word 191
2. PLC Browser Commands 192
2.1 DHCP 193
2.2 DNSNAME 194
2.3 FTPPASSWORD 195
2.4 FTPUSER 196
2.5 GATEWAY 197
2.6 GETDATETIME 198
2.7 IP 199
2.8 IPCFG 200
2.9 IPETH 201
2.10 NETMASK 202
2.11 PING 203
2.12 REBOOT 204
2.13 SETDATETIME 205
2.14 TCPIPMEM 206
2.15 VER 207
2.16 WEBPASSWORD 208
2.17 WEBUSER 209

3. Error handling 210

1. Target libraries

The libraries below provide access to the special hardware and software functionalities of the specific target. They are implemented on the target system and can be used by the CoDeSys project. In simulation mode the target system is not reachable. To allow running a CoDeSys project which uses these libraries in simulation mode, there will be dummy functions called instead. These dummy functions return a neutral value, e.g. "nothing received" for "SysComRecv", "Sending successfull" for "SysComSend".

The following libraries are available:

The RS485/Serial library (rs485.lib)
The SysLibCom library (SysLibCom.lib)
The SysLibSockets library (SysLibSockets.lib)
The RTOS functions library (rtos.lib)
The SysLibEvent library (SysLibEvent.lib)
The SysLibCallback library (SysLibCallback.lib)
The SysLibDir library (SysLibDir.lib)
The SysLibPlcCtrl library (SysLibPlcCtrl.lib)
The SysLibStr library (SysLibStr.lib)
The FRAM library (FRAM_Lib.lib)
The RTC library (RTC_Lib.lib)
The MODEM library (MODEM_Lib.lib)
The TCPIP library (TCPIP_Lib.lib)
The MySQL library (MySQL_Lib.lib)
The MODBUS library (MODBUS_Lib.lib)
The RS485/Expansion library (RS485_Lib.lib)

1.1 The SysLibCom library (SysLibCom.lib)

The implementation of the SysLibCom.lib on the CoDeSys Platform includes the standard CoDeSys SysLibCom functions. For help on these functions is referenced to the standard SysLibCom documentation. Thereby some limitations should be taken into account which are described below.
This chapter also describes the additional functions to the standard CoDeSys SysLibCom library.

Comments

The ports COM1 to COM3 are physical ports. Refer to the user manual of specific target for the availability and type of interface (RS485/RS232).

RS232 port can also be used as programming port. Set COMx=PROG (where x is RS232 port number) in the CHIP.INI file to enable this option. The default for COMx option is USER enabling this port for user application by IEC libraries.
For COM1 to COM3 must be used RS485/Serial library instead or specific protocol library (for example MODBUS RTU).

Limitations

The following limitations have to be taken into account:

- The DTR/DSR/DCD Signals are not supported.
- The option 1 for stop bits (1.5 stop bits) is not supported.
- The total size of bits (data + parity + stop) may not exceed 10 bits.
- It is strongly recomendend to check the return values of the functions.

Here is a list of all the functions this library offers:

SysComFlush
SysComPurge
SysComRxNum
SysComReadLine

1.1.1 SysComFlush

FUNCTION SysComFlush : **BOOL**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Wait until all data from the output buffer has been sent.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle

1.1.2 SysComPurge

FUNCTION SysComPurge : **BOOL**

VAR_INPUT
  dwHandle : *DWORD*;
  bPurgeIn : *BOOL*;
  bPurgeOut : *BOOL*;
END_VAR

Discard the contents of the serial Transmit and/or Receive buffer(s).

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle

bPurgeIn

Set to TRUE to purge the Input buffer

bPurgeOut

Set to TRUE to purge the Output buffer

1.1.3 SysComReadLine

FUNCTION SysComReadLine : **DWORD**

VAR_INPUT
  dwHandle : *DWORD*;
  dwBufferAddress : *DWORD*;
  dwMaxBytesToRead : *DWORD*;
  dwTimeout : *DWORD*;
END_VAR

Read a line from the Com port until Linefeed or max chars.

Return Value

Returns the number of characters read (including linefeed)
0 : Error or no chars read

Input Parameters

dwHandle

Port handle, acquired by SysComOpen

dwBufferAddress

Address of the buffer to store the read chars

dwMaxBytesToRead

Maximum number of characters to read

dwTimeout

Time (ms) after that the function returns at the latest

1.1.4 SysComRxNum

FUNCTION SysComRxNum : **INT**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Return the number of available characters in the input buffer.

Return Value

The number of characters available

Input Parameters

dwHandle

Port handle

1.2 The SysLibSockets library (SysLibSockets.lib)

This chapter describes all functions which are available.

Important: Some functions of the socket interface are blocking. This means that the task which is calling the function pauses until the function can return an expected result (e.g. SysSockRecv waits for some TCP data, SysSockRecvFrom waits for receiving an UDP datagram, SysSockAccept waits for an incoming connection, ...).

This library offers the following functions and data types:

INADDR
SOCK_IP_MREQ
SOCKADDRESS
SOCKET_FD_SET
SOCKET_LINGER
SOCKET_TIMEVAL
SysSockAccept
SysSockBind
SysSockClose
SysSockConnect
SysSockCreate
SysSockGetHostByName
SysSockGetHostName
SysSockGetLastErrorSync
SysSockGetOption
SysSockHtonl
SysSockHtons
SysSockInetAddr
SysSockInetNtoa
SysSockIoctl
SysSockListen
SysSockNtohl
SysSockNtohs
SysSockRecv
SysSockRecvFrom
SysSockSelect
SysSockSend
SysSockSendTo
SysSockSetIPAddress
SysSockSetOption
SysSockShutdown

1.2.1 TYPE INADDR

TYPE INADDR

```
TYPE
    INADDR :
    STRUCT
        S_addr : DWORD;
    END_STRUCT
END_TYPE
```

Contains an IP address.

Member

S_addr

IP address in DWORD format. To compute the IP address in DWORD format see function SysSockInetAddr.

1.2.2 TYPE SOCK_IP_MREQ

TYPE SOCK_IP_MREQ

```
TYPE
    SOCK_IP_MREQ :
    STRUCT
        imr_multiaddr : INADDR;
        imr_interface : INADDR;
    END_STRUCT
END_TYPE
```

Multicast request data type.

Member

imr_multiaddr

Multicast group to join

imr_interface

Interface to join on

1.2.3 TYPE SOCKADDRESS

TYPE SOCKADDRESS

```
TYPE
    SOCKADDRESS :
    STRUCT
        sin_family : INT;
        sin_port : UINT;
        sin_addr : UDINT;
        sin_zero : ARRAY [0..7] OF SINT;
    END_STRUCT
END_TYPE
```

Contains an IP address and a port number.

Member

sin_family

Address family (must by equal to SOCKET_AF_INET)

sin_port

Port in network order. To compute the port in network order use function SysSockHtons.

sin_addr

IP address in DWORD format. To compute the IP address in DWORD format see function SysSockInetAddr.

sin_zero

Not used

1.2.4 TYPE SOCKET_FD_SET

TYPE SOCKET_FD_SET

```
TYPE
    SOCKET_FD_SET :
    STRUCT
        fd_count : UDINT;
        fd_array : ARRAY [0..63] OF DINT;
    END_STRUCT
END_TYPE
```

Contains a socket descriptor set.

Member

fd_count

Number of sockets in this set

fd_array

Socket descriptor list

1.2.5 TYPE SOCKET_LINGER

TYPE SOCKET_LINGER

```
TYPE
    SOCKET_LINGER :
    STRUCT
        l_onoff : WORD;
        l_linger : WORD;
    END_STRUCT
END_TYPE
```

Linger data type of TCP socket

Member

l_onoff

On/Off of linger

l_linger

Value of linger

1.2.6 TYPE SOCKET_TIMEVAL

TYPE SOCKET_TIMEVAL

```
TYPE
    SOCKET_TIMEVAL :
    STRUCT
        tv_sec : DINT;
        tv_usec : DINT;
    END_STRUCT
END_TYPE
```

Contains a Time specification.

Member

tv_sec

Time in Seconds

tv_usec

Time in Microseconds

1.2.7 SysSockAccept

FUNCTION SysSockAccept : **DINT**

VAR_INPUT
  diSocket : ***DINT***;
  pSockAddr : ***POINTER to SOCKADDR***;
  piSockAddrSize : ***POINTER to DINT***;
END_VAR

Accepts an incoming connections.

NOTE: pSockAddr->sin_family must be set to SOCKET_AF_INET prior to call.

Return Value

-1 : error
>0 : success, new Socketdescriptor of accepted connection

Input Parameters

diSocket

Socket descriptor of the listening socket which waits for connections

pSockAddr

Pointer to a SOCKADDRESS struct where port and address of the incoming connection will be stored

piSockAddrSize

Pointer to a variable of type DINT. This variable has got assigned the length of the structure SockAddr (can be retrieved with the aid of the SIZEOF operator).

1.2.8 SysSockBind

FUNCTION SysSockBind : **BOOL**

VAR_INPUT
  diSocket : ***DINT***;
  pSockAddr : ***POINTER to SOCKADDRESS***;
  diSockAddrSize : ***DINT***;
END_VAR

Binds a socket to a specified local port or local address.

NOTE: pSockAddr->sin_family must be set to SOCKET_AF_INET prior to call.

Return Value

TRUE : success
FALSE : error

Input Parameters

diSocket

Socket descriptor

pSockAddr

Pointer to a SOCKADDRESS struct which specifies port and address

diSockAddrSize

Size of the SOCKADDRESS struct in bytes

1.2.9 SysSockClose

FUNCTION SysSockClose : **BOOL**

```
VAR_INPUT
   diSocket : DINT;
END_VAR
```

Closes a opened socket.

Return Value

TRUE : success
FALSE : error

Input Parameters

diSocket

Socket descriptor of the socket which should be closed

1.2.10 SysSockConnect

FUNCTION SysSockConnect : **BOOL**

VAR_INPUT
  diSocket : *DINT*;
  pSockAddr : *POINTER to SOCKADDR*;
  diSockAddrSize : *DINT*;
END_VAR

Establishs a connection to a TCP server.

NOTE: pSockAddr->sin_family must be set to SOCKET_AF_INET prior to call.

Return Value

TRUE : success
FALSE : error

Input Parameters

diSocket

Socket descriptor

pSockAddr

Pointer to a SOCKADDRESS struct which contains the address and the port of the TCP server

diSockAddrSize

Size of the SOCKADDRESS struct in bytes

1.2.11 SysSockCreate

FUNCTION SysSockCreate : **DINT**

```
VAR_INPUT
   diAddressFamiliy : DINT;
   diType : DINT;
   diProtocol : DINT;
END_VAR
```

Opens a new socket for UDP or TCP communication.

Return Value

-1 : error
>0 : success, new socket descriptor

Input Parameters

diAddressFamiliy

Address family, SOCKET_AF_INET supported only

diType

Socket type, SOCKET_STREAM for TCP, SOCKET_DGRAM for UDP

diProtocol

Protocol type, SOCKET_IPPROTO_IP supported only

1.2.12 SysSockGetHostByName

FUNCTION SysSockGetHostByName : **DWORD**

VAR_INPUT
  pHostName : *POINTER to STRING*;
END_VAR

Get the name of the host.

Return Value

SOCKET_INADDR_NONE : error, (16#FFFFFFFF)
else : success, valid hostaddress

Input Parameters

pHostName

Pointer to variable of type STRING containing the hostname

1.2.13 SysSockGetHostName

FUNCTION SysSockGetHostName : **BOOL**

VAR_INPUT
  pHostName : *POINTER to STRING*;
  diNameLength : *DINT*;
END_VAR

Get the name of the host.

Return Value

TRUE : success
FALSE : error

Input Parameters

pHostName

Pointer to a variable of type STRING in which the hostname will be written (minimal 16 chars)

diNameLength

Length of the string in which the hostname will be written (minimal 16)

1.2.14 SysSockGetLastErrorSync

FUNCTION SysSockGetLastErrorSync : **DINT**

```
VAR_INPUT
   diSocket : DINT;
END_VAR
```

Return error code of last socket error.

Return Value

Error code of last occurred socket error

Input Parameters

diSocket

Socket descriptor

Comments

Socket error codes:

201 Operation not permitted
202 No such file or directory
203 No such process
204 Interrupted system call
205 Input/output error
206 Device not configured
209 Bad file descriptor
210 No child processes
211 Cannot allocate memory
213 Permission denied
214 Bad address
217 File exists
219 Operation not supported by device
220 Not a directory
221 Is a directory
222 Invalid argument
224 No resource available
235 Operation would block
236 Operation now in progress
237 Operation already in progress
238 Socket operation on non-socket
239 Destination address required
240 Message too long
241 Protocol wrong type for socket
242 Protocol not available
243 Protocol not supported
244 Socket type not supported
245 Operation not supported
246 Protocol family not supported
247 Address family not supported by protocol family
248 Address already in use
249 Can't assign requested address
250 Network is down
251 Network is unreachable
252 Network dropped connection on reset
253 Software caused connection abort
254 Connection reset by peer
255 No buffer space available
256 Socket is already connected
257 Socket is not connected
258 Can't send/receive after socket shutdown. There is no more data to be received.
259 Too many references: can't splice
260 Operation timed out

261 Connection refused
264 Host is down
265 No route to host

1.2.15 SysSockGetOption

FUNCTION SysSockGetOption : **BOOL**

VAR_INPUT
  diSocket : *DINT*;
  diLevel : *DINT*;
  diOption : *DINT*;
  pOptionValue : *POINTER*;
  piOptionValueLength : *POINTER to DINT*;
END_VAR

Reads the setting of specified socket and the specified option.

Return Value

TRUE : success
FALSE : error

Input Parameters

diSocket

Socket descriptor

diLevel

Protocol level

diOption

Option which should be read (see below)

pOptionValue

Pointer to a variable to store the option value (see below)

piOptionValueLength

Pointer to an DINT which tells the size of the variable to which pOptionValue points (see below)

Comments

Supported Options:

- diLevel = SOCKET_SOL
  - diOption = SOCKET_SO_REUSEADDR
    Value type: INT
    Value: 1 = enable reuse address, 0 = disable reuse address, default: 1
  - diOption = SOCKET_SO_KEEPALIVE
    Value type: INT
    Value: keepalive interval in seconds, min: 10, max: 32767, default: 7200
  - diOption = SOCKET_SO_SNDBUF
    Value type: DINT
    Value: Sendbuffer size for that socket in bytes, default: 4096 for TCP, 2048 for UDP.
    We recommend a maximum size of 8192 bytes.
  - diOption = SOCKET_SO_RCVBUF
    Value type: DINT
    Value: Receivebuffer size for that socket in bytes, default:4096 for TCP, 2048 for UDP.
    We recommend a maximum size of 8192 bytes.
  - diOption = SOCKET_SO_ERROR
    Value type: DUINT
    Value: Last error occurred on that socket (error codes see below)
- diLevel = SOCKET_IPPROTO_TCP
  - diOption = SOCKET_TCP_NODELAY
    Value type: INT
    Value: -1 = disable nagle algorithm, 0 = nagle algorithm is enabled, default: 0

Socket error codes:

201 Operation not permitted
202 No such file or directory
203 No such process
204 Interrupted system call
205 Input/output error
206 Device not configured
209 Bad file descriptor
210 No child processes
211 Cannot allocate memory
213 Permission denied
214 Bad address
217 File exists
219 Operation not supported by device
220 Not a directory
221 Is a directory
222 Invalid argument
224 No resource available
235 Operation would block
236 Operation now in progress
237 Operation already in progress
238 Socket operation on non-socket
239 Destination address required
240 Message too long
241 Protocol wrong type for socket
242 Protocol not available
243 Protocol not supported
244 Socket type not supported
245 Operation not supported
246 Protocol family not supported
247 Address family not supported by protocol family
248 Address already in use
249 Can't assign requested address
250 Network is down
251 Network is unreachable
252 Network dropped connection on reset
253 Software caused connection abort
254 Connection reset by peer
255 No buffer space available
256 Socket is already connected
257 Socket is not connected
258 Can't send after socket shutdown
259 Too many references: can't splice
260 Operation timed out
261 Connection refused
264 Host is down
265 No route to host

1.2.16 SysSockHtonl

FUNCTION SysSockHtonl : **DWORD**

```
VAR_INPUT
   dwHost : DWORD;
END_VAR
```

Converts a DWORD from host byte order to network byte order (in context with Big Endian and Little Endian format).

Return Value

Converted DWORD value

Input Parameters

dwHost

Value to convert

1.2.17 SysSockHtons

FUNCTION SysSockHtons : **WORD**

```
VAR_INPUT
   wHost : WORD;
END_VAR
```

Converts a WORD from host byte order to network byte order (in context with Big Endian and Little Endian format).

Return Value

Converted WORD value

Input Parameters

wHost

Value to convert

1.2.18 SysSockInetAddr

FUNCTION SysSockInetAddr : **DWORD**

```
VAR_INPUT
   stIPAddr : STRING;
END_VAR
```

Converts an IP address from String into a DWORD for the INADDR struct.

Return Value

IP Address as DWORD. This return value will be used to specify the field S_addr of the INADDR struct.

Input Parameters

stIPAddr

IP address string (e.g. '192.168.100.1')

1.2.19 SysSockInetNtoa

FUNCTION SysSockInetNtoa : **BOOL**

VAR_INPUT
  pInAddr : *POINTER to INADDR*;
  pIPAddr : *POINTER to STRING*;
  diIPAddrSize : *DINT*;
END_VAR

Converts a IP address from INADDR format to String.

Return Value

TRUE : success
FALSE : error

Input Parameters

pInAddr

Pointer to structure INADDR which contains the IP address

pIPAddr

Pointer to String where the IP address in String format should be stored

diIPAddrSize

Size of the pIPAddr String

1.2.20 SysSockIoctl

FUNCTION SysSockIoctl : **DINT**

VAR_INPUT
  diSocket : *DINT*;
  diCommand : *DINT*;
  piParameter : *POINTER*;
END_VAR

Manipulates a socket.

Return Value

Depends on the `diCommand` parameter

Input Parameters

diSocket

Socket descriptor

diCommand

Command which specifies what should be done with the socket (see below)

piParameter

Pointer to Parameter, depending on the command (see below)

Comments

Possible commands:

- SOCKET_FIONREAD - Returns the number of bytes waiting in the receive buffer. Returns 0 on success and -1 on error.
  - Parameter: Pointer to DINT where the number of bytes will be stored
- SOCKET_FIONBIO - Switches the socket between nonblocking and blocking mode. (*) Returns 0 on success and -1 on error.
  - Parameter: Pointer to DINT which specifies if the socket should be switched in blocking or in nonblocking mode (0 -> Blocking mode, <>0 -> Nonblocking mode)

(*) By default every socket is in blocking mode so all functions are blocking. This means a socket function does not return until an expected result can be returned. So the calling task pauses until the function returns. E.g. the SysSockAccept function does not return until a client establishes a connection to the socket. Or the SysSockRecv function does not return until some data is available in the receive buffer of the socket. Using the SOCKET_FIONBIO command, a socket (and so its functions) can be switched into the non blocking mode. Using the return values of the socket functions you can decide if the function call was successful or if an error has occured. If an error has occured you can use the function SysSockGetOption to read the error code. With the error code you can see if a real error has occured or the function would block. If the function would block, you have to call that function again until it would not block anymore.

Example in ST:

```
IF NOT ConnectionIsEstablished THEN
  bRet := SysSockConnect(sd, ADR(addr), sizeof(addr));
  IF bRet=FALSE THEN
    diError := 0;
    dwErrorSize := sizeof(diError);
    bRet := SysSockGetOption(sd, SOCKET_SOL, SOCKET_ERROR, ADR(diError),
ADR(diErrorSize));
    IF (bRet=FALSE) OR (diError<>235) THEN
      bConnectionError := TRUE;
    END_IF
  ELSE
```

```
    ConnectionIsEstablished := TRUE;
  END_IF
END_IF
```

1.2.21 SysSockListen

FUNCTION SysSockListen : **BOOL**

VAR_INPUT
  diSocket : ***DINT***;
  diMaxConnections : ***DINT***;
END_VAR

Switches a sockets into the listen mode so it can accept connection requests.

Return Value

TRUE : success
FALSE : error

Input Parameters

diSocket

Socket descriptor

diMaxConnections

Max. number of simultanious connections

1.2.22 SysSockNtohl

FUNCTION SysSockNtohl : **DWORD**

```
VAR_INPUT
   dwNet : DWORD;
END_VAR
```

Converts a DWORD from network byte order to host byte order (in context with Big Endian and Little Endian format).

Return Value

Converted DWORD value

Input Parameters

dwNet

Value to convert

1.2.23 SysSockNtohs

FUNCTION SysSockNtohs : **WORD**

```
VAR_INPUT
   wNet : WORD;
END_VAR
```

Converts a WORD from network byte order to host byte order (in context with Big Endian and Little Endian format).

Return Value

Converted WORD value

Input Parameters

wNet

Value to convert

1.2.24 SysSockRecv

FUNCTION SysSockRecv : **DINT**

```
VAR_INPUT
   diSocket : DINT;
   pbyBuffer : POINTER to BYTE;
   diBufferSize : DINT;
   diFlags : DINT;
END_VAR
```

Receives some data from an established TCP connection.

Return Value

Number of the received bytes

Input Parameters

diSocket

Socket descriptor of the established connection

pbyBuffer

Pointer to buffer of type BYTE where the received data should be stored

diBufferSize

Size of the buffer

diFlags

Flags (must be equal to null)

1.2.25 SysSockRecvFrom

FUNCTION SysSockRecvFrom : **DINT**

VAR_INPUT
  diSocket : ***DINT***;
  pbyBuffer : ***POINTER to BYTE***;
  diBufferSize : ***DINT***;
  diFlags : ***DINT***;
  pSockAddr : ***POINTER to SOCKADRR***;
  diSockAddrSize : ***DINT***;
END_VAR

Receives a UDP datagram from another UDP station.

NOTE: pSockAddr->sin_family must be set to SOCKET_AF_INET prior to call.

Return Value

Number of the received bytes

Input Parameters

diSocket

Socket descriptor

pbyBuffer

Pointer to buffer of type BYTE where the datagram should be stored

diBufferSize

Size of the buffer

diFlags

Flags (must be equal to null)

pSockAddr

Pointer to a SOCKADDRESS struct where the source address and port (of the sender) should be stored

diSockAddrSize

Size of the SOCKADDRESS struct in bytes

1.2.26 SysSockSelect

FUNCTION SysSockSelect : **DINT**

VAR_INPUT
  diWidth : ***DINT***;
  fdRead : ***POINTER to SOCKET_FD_SET***;
  fdWrite : ***POINTER to SOCKET_FD_SET***;
  fdExcept : ***POINTER to SOCKET_FD_SET***;
  ptvTimeout : ***POINTER to SOCKET_TIMEVAL***;
END_VAR

Determine status of one or more sockets.

Return Value

0 : timeout
>0 : number of sockets for which a specified event has occurred

Input Parameters

diWidth

Number of elements of the fd_array field in the structure SOCKET_FD_SET

fdRead

Pointer to structure, defining the socket set from which the read status should be checked

fdWrite

Pointer to structure, defining the socket set from which the write status should be checked

fdExcept

Pointer to structure, defining the socket set from which the Error status should be checked

ptvTimeout

Pointer to structure containing the maximum time to wait for a result

Comments

For any of the pointers to SOCKET_FD_SET, a NULL Pointer is allowed when this group is not relevant.

1.2.27 SysSockSend

FUNCTION SysSockSend : **DINT**

VAR_INPUT
  diSocket : ***DINT***;
  pbyBuffer : ***POINTER to BYTE***;
  diBufferSize : ***DINT***;
  diFlags : ***DINT***;
END_VAR

Sends some data to an established TCP connection.

Return Value

-1 : error
else : success, number of bytes sent

Input Parameters

diSocket

Socket descriptor of the established connection

pbyBuffer

Pointer to buffer of type BYTE which should be sent

diBufferSize

Size of the data which should be sent in bytes

diFlags

Flags (must be equal to null)

1.2.28 SysSockSendTo

FUNCTION SysSockSendTo : **DINT**

VAR_INPUT
  diSocket : ***DINT***;
  pbyBuffer : ***POINTER to BYTE***;
  diBufferSize : ***DINT***;
  diFlags : ***DINT***;
  pSockAddr : ***POINTER to SOCKADDR***;
  diSockAddrSize : ***DINT***;
END_VAR

Sends a UDP datagram to another UDP station.

NOTE: pSockAddr->sin_family must be set to SOCKET_AF_INET prior to call.

Return Value

Number of transfered bytes

Input Parameters

diSocket

Socket descriptor

pbyBuffer

Pointer to buffer of type BYTE which should be sent

diBufferSize

Size of the data to send in bytes

diFlags

Flags (must be equal to null)

pSockAddr

Points to a SOCKADDRESS struct which contains the destination address and port

diSockAddrSize

Size of the SOCKADDRESS struct in bytes

1.2.29 SysSockSetIPAddress

This function is not implemented in this version of the Release.

NOTE: A call to this function returns always zero.

1.2.30 SysSockSetOption

FUNCTION SysSockSetOption : **BOOL**

VAR_INPUT
  diSocket : ***DINT***;
  diLevel : ***DINT***;
  diOption : ***DINT***;
  pOptionValue : ***POINTER***;
  dwOptionValueLength : ***DWORD***;
END_VAR

Controls option settings for specified Socket.

Return Value

TRUE : success
FALSE : error

Input Parameters

diSocket

Socket descriptor

diLevel

Protocol level

diOption

Option which should be set

pOptionValue

Pointer to a variable containing the option value

dwOptionValueLength

Length (in Bytes) of the option value

Comments

For the supported options see function SysSockGetOption.

1.2.31 SysSockShutdown

FUNCTION SysSockShutdown : **BOOL**

VAR_INPUT
  diSocket : ***DINT***;
  diHow : ***DINT***;
END_VAR

Closes a connection.

Return Value

TRUE : success
FALSE : error

Input Parameters

diSocket

Socket descriptor

diHow

Must be equal to null

1.3 The RTOS functions library (rtos.lib)

The RTOS library offers various functions of the underlying Real Time Operating System to the PLC programmer.
These functions include IP configuration, device names and versions, and message exchange functions.
Some of the functions are only useful on controllers supporting parallel execution of DOS EXE programs, others can also be used in pure PLC applications.

Here is a list of all the functions and data type this library offers:

TYPE MSG_EX
RtosCreateMsg
RtosDeleteMsg
RtosDhcpUse
RtosFindMsg
RtosGetBootstrapVersion
RtosGetDevicenames
RtosGetDhcpStat
RtosGetIniEntry
RtosGetIniEntryEx
RtosGetLinkstate
RtosGetMacAddress
RtosGetMsg
RtosGetRebootReason
RtosGetVersion
RtosGetVersionString
RtosIpconfig
RtosReboot
RtosSendMsg
RtosServers
RtosSetIniEntry
RtosSetIniEntryEx

1.3.1 TYPE MSG_EX

TYPE MSG_EX

```
TYPE
    MSG_EX :
    STRUCT
        msgID : UINT := 0;
        name0 : USINT := 77; (* 'M' *)
        name1 : USINT := 83; (* 'S' *)
        name2 : USINT := 71; (* 'G' *)
        name3 : USINT := 88; (* 'X' *)
        mb0 : INT := 2;
        mb1 : INT := 2;
        mb2 : INT := 2;
        mb3 : INT := 2;
    END_STRUCT
END_TYPE
```

This structure contains information about a message exchange. It is used with the message exchange functions from the RTOS library, like

RtosCreateMsg
RtosDeleteMsg
RtosSendMsg
RtosGetMsg
RtosFindMsg

Member

msgID

filled by RtosCreateMsg function

name0

'M'

name1

'S'

name2

'G'

name3

'X'

mb0

number of envelopes, priority 0 (high)

mb1

number of envelopes, priority 1

mb2

number of envelopes, priority 2

mb3

number of envelopes, priority 3 (low)

1.3.2 RtosCreateMsg

FUNCTION RtosCreateMsg : **INT**

VAR_INPUT
  pMsgEx : *POINTER to MSG_EX*;
END_VAR

This function creates a message exchange. You must call this function before the message exchange mechanism can be used. RtosCreateMsg fills in the msgID member of the MSG_EX structure. This ID number of the message exchange will be used by other functions to access the message exchange.
Additionally, you can provide a unique 4 character tag to identify the message exchange when using the RtosFindMsg function.

Return Value

Returns 0 in case of success, otherwise the error code from the operating system

Input Parameters

pMsgEx

Pointer to a MSG_EX structure

Related Topics

RtosDeleteMsg

RtosFindMsg

1.3.3 RtosDeleteMsg

FUNCTION RtosDeleteMsg : **INT**

```
VAR_INPUT
   msgID : UINT;
END_VAR
```

This function deletes a message exchange that is no longer used.

Return Value

Returns 0 in case of success, otherwise an error code

Input Parameters

msgID

Number of the message exchange, as returned by RtosCreateMsg in the structure MSG_EX

Related Topics

RtosCreateMsg

1.3.4 RtosDhcpUse

FUNCTION RtosDhcpUse : **BYTE**

```
VAR_INPUT
   onOffRead : BYTE;
END_VAR
```

Switches DHCP usage on/off or determines wether DHCP usage is on/off.

Return Value

0 = DHCP is off, 1 = DHCP is on

Input Parameters

onOffRead

0 = set DHCP off, 1 = set DHCP on, 2 = read DHCP usage

Related Topics

RtosIpconfig

1.3.5 RtosFindMsg

FUNCTION RtosFindMsg : **INT**

```
VAR_INPUT
   pID : POINTER to UINT;
   pName : POINTER to STRING;
END_VAR
```

Finds the ID of a message exchange by the 4 character name.

Return Value

Returns 0 on success, otherwise error code

Input Parameters

pID

Pointer to a UINT into which the ID of the message exchange will be stored

pName

Pointer to 4 bytes containing the name of the message exchange, as defined in the MSG_EX structure before calling RtosCreateMsg

Related Topics

RtosCreateMsg

1.3.6 RtosGetBootstrapVersion

FUNCTION RtosGetBootstrapVersion : **WORD**

```
VAR_INPUT
   dummy : BYTE;
END_VAR
```

Returns the version of the processor's bootstrap loader.

Return Value

The bootstrap loader version is returned in one WORD, where the high byte is the more significant digit, the low byte is the less significant digit

Input Parameters

dummy

Not used

Related Topics

RtosGetVersion

RtosGetVersionString

1.3.7 RtosGetDevicenames

FUNCTION RtosGetDevicenames : **BYTE**

VAR_INPUT
  ppChipName : ***POINTER to POINTER to STRING***;
  ppIniName : ***POINTER to POINTER to STRING***;
  ppProductName : ***POINTER to POINTER to STRING***;
END_VAR

Returns pointers to the device names of the controller.

Input Parameters

ppChipName

Output paramter: pointer to a string pointer which will receive a pointer to the fixed device name stored in the processor's flash memory

ppIniName

Output paramter: pointer to a string pointer which will receive a pointer to the device name configured in CHIP.INI

ppProductName

Output paramter: pointer to a string pointer which will receive a pointer to the fixed product device name stored in the flash memory

Comments

Example for using RtosGetDeviceNames in Structured Text language:

```
VAR
   p1 : POINTER TO STRING;
   p2 : POINTER TO STRING;
   p3 : POINTER TO STRING;
   s1 : STRING;
   s2 : STRING;
   s3 : STRING;
END_VAR

RtosGetDevicenames(ADR(p1), ADR(p2), ADR(p3));
s1 := p1^;
s2 := p2^;
s3 := p3^;
```

1.3.8 RtosGetDhcpStat

FUNCTION RtosGetDhcpStat : **WORD**

```
VAR_INPUT
   dummy : BYTE;
END_VAR
```

Returns the status of the DHCP client (if DHCP is on).

Return Value

0 = DHCP configuration in progress
1 = successfully configured via DHCP
2 = DHCP failed

Input Parameters

dummy

Not used

Related Topics

RtosDhcpUse

1.3.9 RtosGetIniEntry

FUNCTION RtosGetIniEntry : **INT**

VAR_INPUT
  pSection : *POINTER to STRING*;
  pItemname : *POINTER to STRING*;
  pItemtext : *POINTER to STRING*;
  len : *UINT*;
END_VAR

Reads an entry from the configuration file CHIP.INI on the controller's flash disk.

Return Value

0 = entry not found
-1 = CHIP.INI not found
otherwise number of characters read

Input Parameters

pSection

Pointer to string holding the section

pItemname

Pointer to string holding the name of the desired entry

pItemtext

Pointer to string in which the text of the desired entry will be copied (must be `max_len+1` size)

len

Maximum lenghth of the target string at pItemtext, not including terminating null character

Comments

The functions RtosGetIniEntry and RtosSetIniEntry are not reentrant. Do not use these in different tasks or in combination with other functions or commands which write to CHIP.INI, e.g. the DHCP or IP configuration functions.

Related Topics

RtosGetIniEntryEx

RtosSetIniEntry

RtosSetIniEntryEx

1.3.10 RtosGetIniEntryEx

FUNCTION RtosGetIniEntryEx : **INT**

VAR_INPUT
  pFilename : *POINTER to STRING*;
  pSection : *POINTER to STRING*;
  pItemname : *POINTER to STRING*;
  pItemtext : *POINTER to STRING*;
  len : *UINT*;
END_VAR

Reads an entry from a specified configuration file on the controller's flash disk.

Return Value

0 = entry not found
-1 = configuration file not found
otherwise number of characters read

Input Parameters

pFilename

Pointer to string holding the configuration file name (max. 80 characters)

pSection

Pointer to string holding the section

pItemname

Pointer to string holding the name of the desired entry

pItemtext

Pointer to string in which the text of the desired entry will be copied (must be `max_len+1` size)

len

Maximum lenghth of the target string at pItemtext, not including terminating null character

Comments

The functions RtosGetIniEntryEx and RtosSetIniEntryEx are not reentrant. Do not use these in different tasks or in combination with other functions or commands which write to the configuration file.

Related Topics

RtosGetIniEntry

RtosSetIniEntry

RtosSetIniEntryEx

1.3.11 RtosGetLinkstate

FUNCTION RtosGetLinkstate : **WORD**

```
VAR_INPUT
   dummy : BYTE;
END_VAR
```

Returns the link state of the controller's ethernet interface.

Return Value

bit 0: 1 = link ok, 0 = no link
bit 2: 1 = initialization of the ethernet device failed

Input Parameters

dummy

Not used

Related Topics

RtosGetMacAddress

1.3.12 RtosGetMacAddress

FUNCTION RtosGetMacAddress : **BYTE**

```
VAR_INPUT
   pBuf : POINTER to BYTE;
END_VAR
```

Returns the MAC address of the controller's ethernet interface.

Return Value

Returns always 0

Input Parameters

pBuf

Pointer to buffer for MAC address

Comments

The buffer must provide 6 bytes of space, e.g. be of the type ARRAY[0..5] OF BYTE

1.3.13 RtosGetMsg

FUNCTION RtosGetMsg : **INT**

```
VAR_INPUT
   msgID : UINT;
   pMsg : DWORD;
END_VAR
```

Gets a message from a specified message exchange. Returns immediately if no message is available.

Return Value

Returns 0 on success, otherwise error code

Input Parameters

msgID

Number of the message exchange, as returned by RtosCreateMsg in the structure MSG_EX

pMsg

Pointer to a 12 byte buffer in which the message will be stored. The format of the data can be defined by the application program

Comments

This function returns immediately with code -28 if no message is available.
When more than one message is available at the message exchange, the oldest (FIFO order) message from the highest priority message queue will be reported.

Related Topics

RtosSendMsg

1.3.14 RtosGetRebootReason

FUNCTION RtosGetRebootReason : **BYTE**

```
VAR_INPUT
   dummy : BYTE;
END_VAR
```

Returns the reason why the controller rebooted.

Return Value

Returns 0 if reboot reason is unknown, 3 if controller was rebooted by the watchdog, or 4 if the controller rebooted due to power failure

Input Parameters

dummy

Not used

Comments

This function relies on the retain mechanism which is per default active to save data in case of power failure. So this function can only be used if the controller supports retain variables.

1.3.15 RtosGetVersion

FUNCTION RtosGetVersion : **DWORD**

```
VAR_INPUT
   dummy : BYTE;
END_VAR
```

Returns the version of the operating system.

Return Value

High byte of the low word = more significant digit of the version number
Low byte of the low word = less significant digit of the version number
If the least significant bit of the high word is set, this is a beta version of the operating system

Input Parameters

dummy

Not used

Related Topics

RtosGetVersionString

RtosGetBootstrapVersion

1.3.16 RtosGetVersionString

FUNCTION RtosGetVersionString : **BYTE**

```
VAR_INPUT
   pBuf : POINTER to BYTE;
   len : INT;
END_VAR
```

Returns the version of the operating system as a string.

Return Value

Returns always 0

Input Parameters

pBuf

Pointer to buffer where the version string will be stored

len

Size of buffer at `pBuf`, including space for terminating null character

Related Topics

RtosGetVersion

RtosGetBootstrapVersion

1.3.17 RtosIpconfig

FUNCTION RtosIpconfig : **BYTE**

```
VAR_INPUT
   set : BYTE;
   pIpString : POINTER to STRING;
   pSubString : POINTER to STRING;
   pGatewayString : POINTER to STRING;
END_VAR
```

Sets or reads the controller's IP configuration.

Input Parameters

set

0 to read, 1 to set configuration

pIpString

Output parameter: pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string when reading the configuration or where the IP address will be read from when setting the configuration

pSubString

Output parameter: pointer to a 16 byte memory area where the subnet mask is to be stored as a null terminated string when reading the configuration or where the subnet mask address will be read from when setting the configuration

pGatewayString

Output parameter: pointer to a 16 byte memory area where the gateway address is to be stored as a null terminated string when reading the configuration or where the gateway address will be read from when setting the configuration

Comments

This function is useful if you want to set the IP configuration from the PLC application itself, e.g. using a graphical operator device. However, in most cases the IP configuration will be edited using the PLC Configuration dialog in the programming system.

The strings are null terminated ASCII strings in dotted decimal notation (e.g. 192.168.10.3).
Any of the parameters can be set to 0 if the respective value is not to be set/read.
After setting the configuration, the ethernet interface will automatically be reconfigured.

Important: This function writes to the CHIP.INI file on the controller's flash disk and is not reentrant. Do not use this function in different tasks or programs or in combination with other commands writing to the CHIP.INI.

Related Topics

RtosDhcpUse

1.3.18 RtosReboot

FUNCTION RtosReboot : **BYTE**

```
VAR_INPUT
   dummy : BYTE;
END_VAR
```

Reboots the controller.

Return Value

This function never returns

Input Parameters

dummy

Not used

Comments

While rebooting, the controller stops and does not respond any more. Do not reboot the controller if this could cause danger for human beings, creatures, or material assets.

1.3.19 RtosSendMsg

FUNCTION RtosSendMsg : **INT**

VAR_INPUT
  msgID : *UINT*;
  prio : *INT*;
  pMsg : *POINTER to BYTE*;
END_VAR

Sends a message to the message exchange.

Return Value

Returns 0 on success, otherwise error code

Input Parameters

msgID

Number of the message exchange, as returned by RtosCreateMsg in the structure MSG_EX

prio

Priority, 0..3 where 0 is highest, 3 is lowest priority

pMsg

Pointer to 12 bytes of data which shall be sent to the message exchange. The format of the data can be defined by the application program

Comments

Messages will be reported in message priority order, and from each priority queue in FIFO order.

Related Topics

RtosGetMsg

1.3.20 RtosServers

FUNCTION RtosServers : **BYTE**

```
VAR_INPUT
   server : BYTE;
   stop : BYTE;
END_VAR
```

Starts/Stops the RTOS system servers.

Return Value

Returns always 0

Input Parameters

server

0 = FTP, 1 = telnet, 2 = web server

stop

0 = start server, 1 = stop server

1.3.21 RtosSetIniEntry

FUNCTION RtosSetIniEntry : **INT**

VAR_INPUT
  pSection : *POINTER to STRING*;
  pItemname : *POINTER to STRING*;
  pItemtext : *POINTER to STRING*;
END_VAR

Writes an entry to the configuration file CHIP.INI on the controller's flash disk.

Return Value

0 = ok
-1 = invalid string length

Input Parameters

pSection

Pointer to string holding the section (max. 40 characters)

pItemname

Pointer to string holding the name of the desired entry (max. 40 characters)

pItemtext

Pointer to string holding the text of the entry to be written (max. 128 characters)

Comments

The functions RtosGetIniEntry and RtosSetIniEntry are not reentrant. Do not use these in different tasks or in combination with other functions or commands which write to CHIP.INI, e.g. the DHCP or IP configuration functions.

Related Topics

RtosGetIniEntry

RtosGetIniEntryEx

RtosSetIniEntryEx

1.3.22 RtosSetIniEntryEx

FUNCTION RtosSetIniEntryEx : **INT**

VAR_INPUT
  pFilename : *POINTER to STRING*;
  pSection : *POINTER to STRING*;
  pItemname : *POINTER to STRING*;
  pItemtext : *POINTER to STRING*;
END_VAR

Writes an entry to a specified configuration file on the controller's flash disk.

Return Value

0 = ok
-1 = invalid string length

Input Parameters

pFilename

Pointer to string holding the configuration file name (max. 80 characters)

pSection

Pointer to string holding the section (max. 40 characters)

pItemname

Pointer to string holding the name of the desired entry (max. 40 characters)

pItemtext

Pointer to string holding the text of the entry to be written (max. 128 characters)

Comments

The functions RtosGetIniEntryEx and RtosSetIniEntryEx are not reentrant. Do not use these in different tasks or in combination with other functions or commands which write to the configuration file.

Related Topics

RtosGetIniEntry

RtosGetIniEntryEx

RtosSetIniEntry

1.4 The SysLibEvent library (SysLibEvent.lib)

Here is a list of all the functions this library offers:

SysEventCreate
SysEventDelete
SysEventSet
SysEventWait

1.4.1 SysEventCreate

FUNCTION SysEventCreate : **DWORD**

VAR_INPUT
  pName : *POINTER to STRING*;
END_VAR

Creates a new event with given name.

Return Value

16#FFFFFFFF : error
else : success, handle to new event which should be created

Input Parameters

pName

Pointer to STRING containing the name of the event

1.4.2 SysEventDelete

FUNCTION SysEventDelete : **BOOL**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Deletes an event.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Handle to event which should be deleted

1.4.3 SysEventSet

FUNCTION SysEventSet : **BOOL**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Sets an event.

Return Value

TRUE : success, event is set
FALSE : error

Input Parameters

dwHandle

Handle to event which should be set

1.4.4 SysEventWait

FUNCTION SysEventWait : **BOOL**

VAR_INPUT
  dwHandle : *DWORD*;
  dwTimeout : *DWORD*;
END_VAR

Wait for an event to occur.

Return Value

TRUE : success, event occurred within timeout period
FALSE : error or timeout

Input Parameters

dwHandle

Handle to event which should be waited for

dwTimeout

Timeout (in ms) after which the function will return at the latest

1.5 The SysLibCallback library (SysLibCallback.lib)

The implementation of the SysLibCallback.lib on the CoDeSys Platform is identical with the standard CoDeSys SysLibCallback documentation. So we request you to use that documentation.

Important Note: For a stable operation of the RTS, it is very important that the callback function is defined exactly as described in the 3S online help. Such a function may not have any local variables! If local variables are needed, an empty Wrapper function must be used.

1.6 The SysLibDir library (SysLibDir.lib)

The implementation of the SysLibDir.lib on the CoDeSys Platform is identical with the standard CoDeSys SysLibDir documentation. So we request you to use that documentation.

Comments

The File DateTime field of the DIRECTORY_INFO stucture is not supported yet. Use the function SysFileGetTime to get this information.

1.7 The SysLibPlcCtrl library (SysLibPlcCtrl.lib)

The implementation of the SysLibPlcCtrl.lib on the CoDeSys Platform is identical with the standard CoDeSys SysLibPlcCtrl documentation. So we request you to use that documentation.

Comments

The functions SysShutdownPlc, SysEnableScheduling, and GetPlcLoad are not supported in this Release.

1.8 The SysLibStr library (SysLibStr.lib)

The implementation of the SysLibStr.lib on the CoDeSys Platform is identical with the standard CoDeSys SysLibStr documentation. So we request you to use that documentation.

Comments

The maximum length of strings is limited to 64Kb in this Release.

1.9 The FRAM library (FRAM_Lib.lib)

This library offers the functions for the management of the FRAM memory.
The FRAM memory (Ferroelectric RAM) is a memory that has all the characteristics of the ideal memory. It is in fact retentive to the default of power supply without necessity of batteries, high-speed in writing (in the order of the microseconds against the milliseconds of the EEprom/Flash memory), rewritetable around 10e16 times (against 10e6/10e5 times of the EEprom/Flash) and retainment of data for 45 years. The management of FRAM device is faculty of the programmer. Nevertheless, in the case of IEC retain memory saving into FRAM (option RETAIN=FRAM in the CHIP.INI file) this is not possible because FRAM is directly handled by the operating system.

This library offers the following functions:

FRAM_Read
FRAM_Write

1.9.1 FRAM_Read

FUNCTION FRAM_Read : BOOL

VAR_INPUT
  Address : ***UINT***;
  NumBytes : ***UINT***;
  Buffer : ***DWORD***;
END_VAR

Reads a FRAM memory area.

Return Value

TRUE : success
FALSE : error

Input Parameters

Address

Start address of FRAM memory area (0-8191)

BumBytes

Number of bytes to read

Buffer

Pointer to buffer where to store readed bytes (use ADR() function)

Comments

This function reads an array of bytes from external FRAM device.
External FRAM device has a dimension of 8Kbytes (8192 bytes) and is addressed from 0 to 8191. For the pointer value of a IEC variable to store the readed bytes, ADR() function can be used.

Related Topics

FRAM_Write

1.9.2 FRAM_Write

FUNCTION FRAM_Write : BOOL

VAR_INPUT
  Address : *UINT*;
  NumBytes : *UINT*;
  Buffer : *DWORD*;
END_VAR

Writes a FRAM memory area.

Return Value

TRUE : success
FALSE : error

Input Parameters

Address

Start address of FRAM memory area (0-8192)

BumBytes

Number of bytes to read

Buffer

Pointer to buffer where to store readed bytes (use ADR() function)

Comments

This function writes an array of bytes into external FRAM device.
External FRAM device has a dimension of 8Kbytes (8192 bytes) and is addressed from 0 to 8191. For the pointer value of a IEC variable to get the bytes to write, ADR() function can be used.

Related Topics

FRAM_Read

1.10 The RTC library (RTC_Lib.lib)

This library offers the functions for the management of the Real Time Clock.

Here is a list of all the functions this library offers:

DateTime_Read
RTC_Read
RTC_Write

1.10.1 DateTime_Read

FUNCTION DateTime_Read : DT

```
VAR_INPUT
   Enable : BOOL;
END_VAR
```

Reads current date and time from Real Time Clock.

Return Value

Returns current date and time into DT type variable

Input Parameters

Enable

Enable the function execution

Comments

This function reads date and time from external Real Time Clock device.
The value returned is of DT (DATE AND TIME) standard type.
You advises to periodically execute the function, for example at intervals of 1s, to refresh the value of date and time into a global variable.
If the function is not enabled returns the reference date and time value (DT#1970-01-01-00:00).

1.10.2 RTC_Read

FUNCTION RTC_Read : SystemTimeDate

```
VAR_INPUT
   Enable : BOOL;
END_VAR
```

Reads current date and time from Real Time Clock.

Return Value

Returns current date and time into SystemTimeDate type variable

Input Parameters

Enable

Enable the function execution

Comments

This function reads date and time from external Real Time Clock device.
The value returned is of SystemTimeDate type defined into SysLibTime.lib standard library.
You advises to periodically execute the function, for example at intervals of 1s, to refresh the value of date and time into a global variable.

Related Topics

RTC_Write

1.10.3 RTC_Write

FUNCTION RTC_Write : BOOL

VAR_INPUT
  Enable : *BOOL*;
  Setup : *SystemTimeDate*;
END_VAR

Updates current date and time into Real Time Clock.

Return Value

TRUE : Success
FALSE : Error

Input Parameters

Enable

Enable the function execution

Setup

Value of SystemTimeDate type to be forced into Real Time Clock

Comments

This function updates date and time into external Real Time Clock device.
The value to be forced is of SystemTimeDate type defined into SysLibTime.lib standard library.

Related Topics

RTC_Read

1.11 The MODEM library (MODEM_Lib.lib)

This library offers the functions for the management of the MODEM .
Refer to the documentation of the specific Target to check the availability and type of modem.

Here is a list of all the functions this library offers:

TYPE DYNDNS_STATUS
TYPE MODEM_CHAR_SET
TYPE MODEM_COMMAND
TYPE MODEM_DATA_SERVICE
TYPE MODEM_STATUS
DynDNS_Client
GM01_PPP
GM01_PPP_SMS_Call
GM01_SMS

NOTE: GM01_PPP and GM01_SMS function blocks perform the functions limited to their type of service and cannot be used simultaneously. For a complete management of all the capabilities of the modem, it's recommend the use of the GM01_PPP_SMS_Call function block which also includes several other features.

1.11.1 DYNDNS_STATUS

TYPE DYNDNS_STATUS

```
TYPE DYNDNS_STATUS :
(
    DYNDNS_STANDBY := 0,
    DYNDNS_UPDATING := 1,
    DYNDNS_UPDATED := 2,
    DYNDNS_FAILED := 3,
    DYNDNS_RETRY_WAIT := 4
) := DYNDNS_STANDBY;
END_TYPE
```

This type enumerates the current status of Dynamic DNS Client.

Member

DYNDNS_STANDBY := 0

Service is waiting for activation

DYNDNS_UPDATING := 1

Updating request in progress

DYNDNS_UPDATED := 2

IP address is updated on the server

DYNDNS_FAILED := 3

Update on the server failed

DYNDNS_RETRY_WAIT := 4

Waiting before retry

1.11.2 MODEM_CHAR_SET

TYPE MODEM_CHAR_SET

```
TYPE MODEM_CHAR_SET :
(
    MODEM_CHAR_SET_ASCII := 0,
    MODEM_CHAR_SET_GSM := 1,
    MODEM_CHAR_SET_UCS2 := 2,
    MODEM_CHAR_SET_UTF8 := 3,
    MODEM_CHAR_SET_8859_1 := 4
) := MODEM_CHAR_SET_ASCII;
END_TYPE
```

This type enumerates the current character set for SMS.

Member

MODEM_CHAR_SET_ASCII := 0

ASCII characters (0x00-0x7F)

MODEM_CHAR_SET_GSM := 1

GSM default alphabet (GSM 03.38 subclause 6.2.1)

MODEM_CHAR_SET_UCS2 := 2

Unicode (ISO/IEC 10646[32])

MODEM_CHAR_SET_UTF8 := 3

Unicode 8 bit (ISO 10646 transformation format)

MODEM_CHAR_SET_8859_1 := 4

Latin 1 (ISO 8859-1)

1.11.3 MODEM_COMMAND

TYPE MODEM_COMMAND

```
TYPE MODEM_COMMAND :
(
    MODEM_NO_COMMAND := 0,
    MODEM_SIGNAL_READ := 1
) := MODEM_NO_COMMAND;
END_TYPE
```

This type enumerates the codes of auxiliary commands for the modem.

Member

MODEM_NO_COMMAND := 0

No command is requested

MODEM_SIGNAL_READ := 1

Update the reading of field strength of the radio network

1.11.4 MODEM_DATA_SERVICE

TYPE MODEM_DATA_SERVICE

```
TYPE MODEM_DATA_SERVICE :
(
    MODEM_DATA_NO := 0,
    MODEM_DATA_GPRS := 1
    MODEM_DATA_EDGE := 2
    MODEM_DATA_UMTS := 3
    MODEM_DATA_UMTS_HSDPA := 4
    MODEM_DATA_UMTS_HSUPA := 5
    MODEM_DATA_UMTS_HSDPA_HSUPA := 6
) := MODEM_DATA_NO;
END_TYPE
```

This type enumerates the codes of packet data service established.

Member

MODEM_DATA_NO := 0

No data service available

MODEM_DATA_GPRS := 1

GPRS data service available

MODEM_DATA_EDGE := 2

EDGE data service available

MODEM_DATA_UMTS := 3

UMTS data service available

MODEM_DATA_UMTS_HSDPA := 4

UMTS/HSDPA data service available

MODEM_DATA_UMTS_HSUPA := 5

UMTS/HSUPA data service available

MODEM_DATA_UMTS_HSDPA_HSUPA := 6

UMTS/HSDPA-HSUPA data service available

1.11.5 MODEM_STATUS

TYPE MODEM_STATUS

```
TYPE MODEM_STATUS :
(
    MODEM_STANDBY := 0,
    MODEM_INIT := 1,
    NET_REGISTRATION := 2,
    PPP_CONNECTING := 3,
    MODEM_CONNECTED := 4,
    PPP_LINK_LOST := 5,
    MODEM_DCD_LOST := 6,
    MODEM_DEINIT := 7,
    MODEM_RETRY_WAIT := 8,
    DYNDNS_REQUEST := 10,
    DYNDNS_OK := 11,
    DYNDNS_FAIL := 12,
    SMS_SENDING := 20,
    SMS_SENDED := 21,
    SMS_SEND_ERROR := 22,
    SMS_GETTING := 23,
    SMS_RECEIVED := 24,
    SMS_NOT_RECEIVED := 25,
    CALL_SENDING := 30,
    CALL_SENDED := 31,
    CALL_SEND_ERROR := 32,
    CALL_RECEIVING := 33,
    CALL_TERMINATED := 34,
    MODEM_SIGNAL_READING := 40
) := MODEM_STANDBY;
END_TYPE
```

This type enumerates the current status of generic MODEM function block.

Member

MODEM_STANDBY := 0

Modem is OFF in standby mode

MODEM_INIT := 1

Initialization of modem connection

NET_REGISTRATION := 2

Registration on the radio network

PPP_CONNECTING := 3

PPP connection in progress

MODEM_CONNECTED := 4

Modem connected to the service

PPP_LINK_LOST := 5

Loss of the PPP link

MODEM_DCD_LOST := 6

Loss of DCD signal of the modem

MODEM_DEINIT := 7

Deinitialization of the modem

MODEM_RETRY_WAIT := 8

Waiting before reconnection

DYNDNS_REQUEST := 10

Updating request in progress

DYNDNS_OK := 11

IP address is updated on the server

DYNDNS_fail := 12

Update on the server failed

SMS_SENDING := 20

SMS sending in progress

SMS_SENDED := 21

SMS sent successfully

SMS_SEND_ERROR := 22

Error while sending the SMS

SMS_GETTING := 23

SMS getting in progress

SMS_RECEIVED := 24

Received and getted an SMS

SMS_NOT_RECEIVED := 25

No SMS available in reception

CALL_SENDING := 30

Outgoing call in progress

CALL_SENDED := 31

Called device has answered to outgoing call

CALL_SEND_ERROR := 32

Error while sending outgoing call

CALL_RECEIVING := 33

Incoming call in progress

CALL_TERMINATED := 34

Call has been terminated by caller/called (hang-up)

MODEM_SIGNAL_READING := 40

Updating the field strength of the radio network

1.11.6 DynDNS_Client

FUNCTION BLOCK DynDNS_Client

```
VAR_INPUT
   Start : BOOL;
   Server_Url : STRING(80);
   User : STRING(20);
   Password : STRING(20);
   Hostname : STRING(50);
END_VAR

VAR_OUTPUT
   Updated : BOOL;
   Status : DYNDNS_STATUS;
   Last_IP : STRING(15);
END_VAR

VAR
   State : INT;
END_VAR
```

Function block of Dynamic DNS client for updating the IP address on Dynamic DNS server.

Input Variables

Start

Activation signal. If TRUE continuously checks if an IP update is required

Server_Url

Url of Dynamic DNS service. For dyndns.com is 'members.dyndns.org/nic/update'

User

Username of the account for the service of Dynamic DNS server

Password

Password of the account for the service of Dynamic DNS server

Hostname

Address of the connected device. Example: 'myplc.dyndns.org'

Output Variables

Updated

Result of the latest update

Status

Current status of the update

Last_IP

Last IP address registered into Dynamic DNS server

Internal Variables

State

Current status of the state graph for the function block

Comments

This function block is maintaining its IP on a Dynamic DNS server as "dyndns.com", "no-ip.com" and others. After establishing a connection with the Internet provider, it usually gives to the device a dynamic IP address that can vary each time.
To simplify access to the system from the outside, through the Internet, it's possible to communicate the current IP to a Dynamic DNS server in order to make known to the whole network. In addition, the server performs an important function associatiating the IP number to a symbolic name chosen for the device and easier to remember. In this way you can access the Web-server and FTP-server of the system by typing the same address string, usually a third-level domain (for example: myplc.dyndns.org).
To use this service you must register, free of charge or with a small amount, at a provider of Dynamic DNS server, choosing a username and password. After creating an account you must install on the server a Hostname (a specific equipment for which perform the service). The Hostname coincides with the full address for the selected device. At this point the function block installs the Dynamic DNS client which must be suitably configured to communicate with the server.
Server_Url parameter is the address where the server is available and this is characteristic of the service provider.
The User and Password are those chosen during registration to the service that usually involves managing of more Hosts.
Updating the IP in the server record, associated with the specific Host, is allowed only when the value of IP changes. For this, while maintaining the function block permanently enabled, it will check continuously the variation of the current IP compared to the value previously stored. Only in case of variation will provide a new request to the server to update its IP.
For each parameter not explicitly defined in the function block it uses the value in the CHIP.INI configuration file and, if not available in that file, it uses a fixed default value:

```
Server_Url    members.dyndns.org/nic/update
User          myplc_user
Password      myplc_password
Hostname      myplc.dyndns.org
(myplc is the [DEVICE]NAME parameter value of CHIP.INI file)
```

The function block returns the current status of the update with one of the values defined by the DYNDNS_STATUS type:

```
Status = 0    Service is waiting for activation
Status = 1    Updating request in progress
Status = 2    IP address is updated on the server
Status = 3    Update on the server failed
Status = 4    Waiting before retry
```

The Last_IP output indicates the value of the last IP address permanently saved in CHIP.INI file.

Related Topics

DYNDNS_STATUS

1.11.7 GM01_PPP

FUNCTION BLOCK GM01_PPP

```
VAR_INPUT
   Start : BOOL;
   Sim_Pin : STRING(12);
   Auth : SINT;
   User : STRING(20);
   Password : STRING(20);
   Connect_String : STRING(80);
   Dial : STRING(20);
   Retry_Seconds : INT;
END_VAR

VAR_OUTPUT
   Ready : BOOL;
   Status : MODEM_STATUS;
   Signal : USINT;
END_VAR

VAR
   State : INT;
   Data_Service : MODEM_DATA_SERVICE;
   My_IP : STRING(15);
   Remote_IP : STRING(15);
   Netmask : STRING(15);
   DNS1 : STRING(15);
   DNS2 : STRING(15);
   Retry_Timer : INT;
END_VAR
```

Function block to establish a permanent Internet connection with PPP for GM01 modem.

Input Variables

Start

Activation signal. If FALSE starts the shutdown procedure

Sim_Pin

Pin code of the data SIM. Default is disabled

Auth

Authorization type: -1=undefined, 0=No, 1=PAP, 2=CHAP, 3=PAPPEER, 4=CHAPPEER

User

Username for the PPP connection. Required if Auth > 0

Password

Password for the PPP connection. Required if Auth > 0

Connect_String

Connection string with the APN. Example: 'AT+CGDCONT=1,"IP","ibox.tim.it"'

Dial

Call string with the dial number. Example: 'ATD*99***1#'

Retry_Seconds

Number of seconds before retrying the connection (-1=undefined)

Output Variables

Ready

Connection ready flag. The Internet link is available

Status

Current status of the connection

Signal

Field strength of the radio network: 0=Min, 10=Max

Internal Variables

State

Current status of the state graph of the function block

Data_Service

Information about the Packet Data Service type of the connection

My_IP

IP address assigned by service provider

Remote_IP

Remote IP address of the peer to peer connection

Netmask

Netmask value for the subnet

DNS1

IP address of DNS server 1

DNS2

IP address of DNS server 2

Retry_Timer

Remaining time to the reconnection

Comments

This function block provides a permanent Internet connection using PPP with a modem of GM01 type. Refer to the documentation of the target to verify the type of modem available.
The function block, once activated with the Start signal, starts a connection procedure that could, during certain phases, block the execution of the POU for relatively long periods. When the connection is established shall enter into a permanent state with a minimum processing time. For this reason, the function block must be inserted into a POU of the IEC program updated by a dedicated task.
When the connection is established, the Ready output becomes TRUE and, from this moment, should be considered the execution of other parts of the POU such as the activation of DynDNS_Client function block.
By setting the Start signal to FALSE, shall start a procedure for disabling the modem connection and shutdown.
The configuration parameters, passed as input to the function block, are not all necessary for the connection. Refer to specific settings of the service provider. Moreover, for each parameter that is not explicitly defined in the function block, the value in the CHIP.INI configuration file is used and, if not available in that file, it uses a fixed default value:

```
Sim_Pin          disabled
Auth             0=No
User             disabled
Password         disabled
Connect_String   AT+CGDCONT=1,"IP","ibox.tim.it"
Dial             ATD*99***1#
```

```
Retry_Seconds    60
```

The function block returns the current state of the connection with one of the values defined by the MODEM_STATUS type:

```
Status = 0    Modem is OFF in standby mode
Status = 1    Initialization of modem connection
Status = 2    Registration on the network
Status = 3    PPP connection in progress
Status = 4    Modem connected to the service
Status = 5    Loss of the PPP link
Status = 6    Loss of DCD signal of the modem
Status = 7    Deinitialization of the modem
Status = 8    Waiting before reconnection
```

The Signal output indicates the field strength of the radio network.

NOTE: permanent PPP connection is useable as an alternative to other types of services like SMS management.

Related Topics

MODEM_DATA_SERVICE

MODEM_STATUS

DynDNS_Client

1.11.8 GM01_PPP_SMS_Call

FUNCTION BLOCK GM01_PPP_SMS_Call

```
VAR_INPUT
   Start : BOOL;
   PPP_Disable : BOOL;
   DynDNS_Enable : BOOL;
   Retry_Seconds : INT;
   Char_Set : MODEM_CHAR_SET;
   SMS_Send_Start : BOOL;
   SMS_Send_Phone : STRING(20);
   SMS_Send_Text : STRING(160);
   SMS_Get_Start : BOOL;
   Call_Send_Start : BOOL;
   Call_Send_Phone : STRING(20);
   Call_Hang_Up : BOOL;
   Command : MODEM_COMMAND;
   Command_Data : STRING(300);
END_VAR

VAR_OUTPUT
   Ready : BOOL;
   Status : MODEM_STATUS;
   Signal : USINT;
   SMS_News : UINT;
   SMS_Get_Time : STRING(32);
   SMS_Get_Phone : STRING(20);
   SMS_Get_Text : STRING(1536);
   Call_Active : BOOL;
   Call_ID_Phone : STRING(20);
   Call_Rings : UINT;
   Command_Replay : STRING(300);
END_VAR

VAR
   State : INT;
   Data_Service : MODEM_DATA_SERVICE;
   My_IP : STRING(15);
   Remote_IP : STRING(15);
   Netmask : STRING(15);
   DNS1 : STRING(15);
   DNS2 : STRING(15);
   Last_IP : STRING(15);
   Retry_Timer : INT;
   Retry_Counter : UDINT;
END_VAR
```

Function block to establish a permanent Internet connection with PPP for GM01 modem. Within the same radio connection all other services like SMS and incoming/outgoing calls are allowable.

Input Variables

Start

Activation signal. If FALSE starts the shutdown procedure

PPP_Disable

Disable the PPP data connection allowing only SMS and Calls

DynDNS_Enable

Enable the DynDNS client function

Retry_Seconds

Number of seconds before retrying the connection (-1=undefined)

Char_Set

Characters set used in text messages according to the MODEM_CHAR_SET type

SMS_Send_Start

Activation signal for sending SMS

SMS_Send_Phone

Phone number to send the SMS

SMS_Send_Text

Text of the SMS to be sent

SMS_Get_Start

Activation signal for getting SMS

Call_Send_Start

Activation signal for sending an outgoing call

Call_Send_Phone

Phone number of outgoing call

Call_Hang_Up

Signal for incoming/outgoing call hang up

Command

Activation of a specific request to modem

Command_Data

Optional data for a specific request to modem

Output Variables

Ready

Connection ready flag. The Internet link is available

Status

Current status of the connection

Signal

Field strength of the radio network: 0=Min, 10=Max

SMS_News

Number of new SMS available

SMS_Get_Time

Time of receipt of the SMS

SMS_Get_Phone

Telephone number of the sender of the SMS

SMS_Get_Text

Text of the SMS received

Call_Active

An incoming call is active (bell is ringing)

Call_ID_Phone

Telephone number of the caller (ID)

Call_Rings

Number of rings detected for the incoming call

Command_Replay

Modem replay to a specific command code

Internal Variables

State

Current status of the state graph of the function block

Data_Service

Information about the Packet Data Service type of the connection

My_IP

IP address assigned by service provider

Remote_IP

Remote IP address of the peer to peer connection

Netmask

Netmask value for the subnet

DNS1

IP address of DNS server 1

DNS2

IP address of DNS server 2

Last_IP

Currently registered IP address on DynDNS server

Retry_Timer

Remaining time to the reconnection

Retry_Counter

Counting of connection retries

Comments

This function block combines, in a single central control, all the operations carried out by a modem of GM01 type. Refer to the documentation of the target to verify the type of modem available.

The function block, once activated with the Start signal, starts a connection procedure that could, during certain phases, block the execution of the POU for relatively long periods. When the connection is established shall enter into a permanent state with a minimum processing time. For this reason, the function block must be inserted into a POU of the IEC program updated by a dedicated task.

The Ready output signal indicates that the modem is connected and the Status output variable assumes the MODEM_CONNECTED value.
The Signal output variable indicates the field strength of the radio connection.
By setting the Start signal to FALSE, shall start a procedure for disabling the modem connection and shutdown.

The PPP data connection is set by default but can be disabled forcing PPP_Disable input signal to TRUE value. This allows the use of function block only for SMS and calls.
If the data connection is enabled, the default address of device gateway is redefined using the IP value provided by the network operator, allowing a permanent radio connection to the Internet.
If DynDNS_Enable signal is active, as soon as the PPP connection is established, a new update of IP address on DynDNS server is also executed.

When the modem is connected, the SMS send/receive services can be executed.
The SMS_Send_Start input signal forces the graph of the function block in the state of sending the SMS according to the previously prepared data. The result of the operation is indicated by the value of the Status output variable. The SMS_Send_Start signal can be forced also for only one cycle and should be resetted to return to idle state waiting for a new command.
The SMS_Get_Start input signal forces the graph of the function block in the state of the received SMS extraction. The Status output variable indicates the availability or not of an SMS in reception and the data of a new SMS are available in the respective output variables. The SMS_Get_Start signal can be forced also for only one cycle and should be resetted to return to idle state waiting for a new command.
The SMS_News output variable indicates the current availability of new incoming messages.

In the state of modem connected the monitoring of any incoming call is always active.
During the entire duration of the incoming call, the Call_Active signal is TRUE, indicating the bell ringing. If the service is enabled to recognize the caller ID, its telephone number is forced into the Call_ID_Phone output variable. The number of rings received is counted in the Call_Rings output variable. When the caller hangs up the call, Call_Active output returns to FALSE, leaving in memory the phone number and the rings counter until the next call. To terminate an incoming call, before the caller hung up, activate, even for a single cycle, the Call_Hang_Up input variable and then reset it inactive.

To make an outgoing call should be set up phone number in the Call_Send_Phone input variable and activate the Call_Send_Start signal.
If the remote device answers the call, the status of the graph goes to Call_SENDED. If the called device hangs up the call or does not answer or is busy, the state, after the transition to the CALL_TERMINATED value, return to MODEM_CONNECTED.
To terminate the outgoing call by the caller, the Call_Hang_Up input signal must be activated. Both Call_Send_Phone and Call_Hang_Up variables must be activated for at least one cycle and then reforced to inactive value FALSE.

Finally in the state of service activated, also some specific modem functions can be executed forcing a command code in the Command input variable. The Command_Data input variable can supply also an optional data string specific for the command type. Any information replayed by the modem is shown in the Command_Replay output variable. For example, the MODEM_SIGNAL_READ command code allows the updating of Signal output value that indicates the signal strength of the radio network.

NOTE: all the configuration parameters of the modem, not explicitly defined in the function block, are taken from the CHIP.INI file.

Related Topics

MODEM_CHAR_SET

MODEM_COMMAND

MODEM_DATA_SERVICE

MODEM_STATUS

1.11.9 GM01_SMS

FUNCTION BLOCK GM01_SMS

```
VAR_INPUT
   Start : BOOL;
   Sim_Pin : STRING(12);
   Char_Set : MODEM_CHAR_SET;
   SMS_Send_Start : BOOL;
   SMS_Send_Phone : STRING(20);
   SMS_Send_Text : STRING(160);
   SMS_Get_Start : BOOL;
END_VAR

VAR_OUTPUT
   Ready : BOOL;
   Status : MODEM_STATUS;
   Signal : USINT;
   SMS_Get_Time : STRING(32);
   SMS_Get_Phone : STRING(20);
   SMS_Get_Text : STRING(1536);
END_VAR

VAR
   State : INT;
END_VAR
```

Function block for handling SMS send/receive with GM01 modem.

Input Variables

Start

Activation signal. If FALSE starts the shutdown procedure

Sim_Pin

Pin code of the SIM. Default is disabled

Char_Set

Characters set used in text messages according to the MODEM_CHAR_SET type

SMS_Send_Start

Activation signal for sending SMS

SMS_Send_Phone

Phone number to send the SMS

SMS_Send_Text

Text of the SMS to be sent

SMS_Get_Start

Activation signal for getting SMS

Output Variables

Ready

SMS service ready flag

Status

Current status of the SMS management

Signal

Field strength of the radio network: 0=Min, 10=Max

SMS_Get_Time

Time of receipt of the SMS

SMS_Get_Phone

Telephone number of the sender of the SMS

SMS_Get_Text

Text of the SMS received

Internal Variables

State

Current status of the state graph of the function block

Comments

This function block allows the management of SMS send/receive using a modem of GM01 type. Refer to the documentation of the target to verify the type of modem available.
The function block, once activated with the Start signal, starts a connection procedure that could, during certain phases, block the execution of the POU for relatively long periods. When the connection is established shall enter into a permanent state with a minimum processing time. For this reason, the function block must be inserted into a POU of the IEC program updated by a dedicated task.
When the SMS service is activated, the Ready output becomes TRUE and, from this moment, should be considered the execution of other parts of the POU. By setting the Start signal to FALSE, shall start a procedure for disabling the modem connection and shutdown.
If the Sim_Pin configuration parameter is not explicitly defined in the function block, the value in the CHIP.INI configuration file is used and, if not available in that file, it uses the disabled value.

The function block returns the current state of the service with one of the values defined by the MODEM_STATUS type:

```
Status = 0    Modem is OFF in standby mode
Status = 1    Initialization of modem connection
Status = 4    Modem connected to the service
Status = 7    Deinitialization of the modem
Status = 20   SMS sending in progress
Status = 21   SMS sent successfully
Status = 22   Error while sending the SMS
Status = 23   SMS getting in progress
Status = 24   Received and getted an SMS
Status = 25   No SMS available in reception
```

When this function block is in the state of the activated service, it awaits the request of a command to send SMS or extract a received SMS. The SMS_Send_Start input signal forces the graph of the block in the state of sending the SMS according to the previously prepared data. The result of the operation is indicated by the value of the Status output variable. Then the SMS_Send_Start signal must be put back inactive to return in the idle state waiting for a new command.
The SMS_Get_Start input signal force the graph of the block in the state of the received SMS extraction. The Status output variable indicates the availability or not of an SMS in reception and the data of a new SMS are available in the respective output variables. Then the SMS_Get_Start signal must be put back inactive to return in the idle state waiting for a new command. To receive more incoming SMS the SMS_Get_Start signal must be activated periodically checking the reception via the Status output value.
The Signal output indicates the field strength of the radio network.

NOTE: SMS management is useable as an alternative to other types of services like permanent PPP connection.

Related Topics

MODEM_CHAR_SET

MODEM_STATUS

1.12 The TCPIP library (TCPIP_Lib.lib)

This library offers the functions for the TCPIP applications.

Here is a list of all the functions this library offers:

HTTP_Get
HTTP_Post
SendMail
SNTP_DateTime

1.12.1 HTTP_Get

FUNCTION HTTP_Get : INT

VAR_INPUT
  Request : *STRING*;
  Result : *STRING*;
  Result_Max_Len : *INT*;
  DNS : *STRING*;
END_VAR

Sends HTTP GET request and receive the result page.

Return Value

0 : success
-1...-4: DNS client error
201...265: TCP/IP functions error
1/2/4/8/16: SSL functions error

Input Parameters

Request

URL of the requested resource (max 2000 characters)

Result

String to receive the response (max 4000 characters)

Result_Max_Len

Max number of characters to receive

DNS

Specific DNS server address (xxx.xxx.xxx.xxx)

Comments

This function sends a request to a web server using an URL with GET method and returns all the string (including header) of the page. The Request parameter is the string containing the request with the form:

```
Username:Password@Host:Port/Path?Query
```

```
Username:Password@   send optional authorization fields
Host                 address of the server that has the resource
:Port                select a port other than the default (80)
/Path                full path to the server resource
?Query               parameters to send: ?par1=xxx&par2=yyy&par3=zzz...
```

The Result parameter is the string in which to insert the response from the server. This response contains a header, typical of HTTP, followed by the actual output of the server (body). The header and the body are separated by a blank line, which can be searched as the beginning of the body part. The Result_Max_Len parameter is the maximum number of characters to receive in the response string.
The DNS parameter allows the specification of a different DNS server which has priority over the default setting into CHIP.INI configuration file.
The SSL connections require an enabling and configuration using the file CHIP.INI (refer to the administration pages).

Example

```
Err_Code := HTTP_Get('www.mysite.com/page.php?par1=123&par2=456', Result, 500, '');
```

Related Topics

HTTP_Post

1.12.2 HTTP_Post

FUNCTION HTTP_Post : INT

VAR_INPUT
  Request : *STRING*;
  Query : *STRING*;
  Result : *STRING*;
  Result_Max_Len : *INT*;
  DNS : *STRING*;
END_VAR

Sends HTTP POST request and receive the result page.

Return Value

0 : success
-1...-4: DNS client error
201...265: TCP/IP functions error
1/2/4/8/16: SSL functions error

Input Parameters

Request

URL of the requested resource (max 400 characters)

Query

String with the parameters list (max 2000 characters)

Result

String to receive the response (max 4000 characters)

Result_Max_Len

Max number of characters to receive

DNS

Specific DNS server address (xxx.xxx.xxx.xxx)

Comments

This function sends a request to a web server using an URL with POST method and returns all the string (including header) of the page. The Request parameter is the string containing the request with the form:

```
Username:Password@Host:Port/Path
```

```
Username:Password@   send optional authorization fields
Host                 address of the server that has the resource
:Port                select a port other than the default (80)
/Path                full path to the server resource
```

Query parameter is the string containing the parameters:

```
Query                parameters to send: par1=xxx&par2=yyy&par3=zzz...
```

The Result parameter is the string in which to insert the response from the server. This response contains a header, typical of HTTP, followed by the actual output of the server (body). The header and the body are separated by a blank line, which can be searched as the beginning of the body part. The Result_Max_Len parameter is the maximum number of characters to receive in the response string.
The DNS parameter allows the specification of a different DNS server which has priority over the default setting into CHIP.INI

configuration file.
The SSL connections require an enabling and configuration using the file CHIP.INI (refer to the administration pages).

Example

```
Err_Code := HTTP_Post('www.mysite.com/page.php', 'par1=123&par2=456', Result, 500,
'');
```

Related Topics

HTTP_Get

1.12.3 SendMail

FUNCTION SendMail : BOOL

```
VAR_INPUT
   Local_IP : STRING;
   From_Address : STRING;
   From_Alias : STRING;
   Server_Add : STRING;
   Username : STRING;
   Password : STRING;
   To_Address : STRING;
   Subject : STRING;
   Attached_File : STRING;
   Text : STRING;
END_VAR
```

Sends an E-mail by the specified SMTP server.

Return Value

TRUE : success
FALSE : error

Input Parameters

Local_IP

IP address of sender into Local Area Network (not necessary)

From_Address

E-mail address of sender

From_Alias

Alias name of sender (not necessary)

Server_Add

Address of SMTP server (E-mail account)

Username

Username for SMTP server login (E-mail account)

Password

Password for SMTP server login (E-mail account)

To_Address

Address of the recipient of E-mail

Subject

Description of the E-mail subject (not necessary)

Attached_File

Path/name of attached file (not necessary)

Text

Text of the E-mail (not necessary)

Comments

This function connects to a SMTP server and sends an E-mail to a destination address.
The first six parameters are normally fixed because they define the sender informations and the specific E-mail account on a SMTP server. The data of the account are furnished by the provider of E-mail service.
Local_IP, From_Address and From_Alias parameters identify the sender, that is the IEC system calling this function to send mail. For the purposes of sending mail are not necessary but however their definition allows the recipient to know the origin of the message.
Server_Add, Username and Password parameters are instead mandatory because they must match those of the E-mail account that is activated by a SMTP service provider.
The Server_Add parameter can be specified in the format xxx.xxx.xxx.xxx or by the server name. In this case the DNS client is used to resolve the IP address of the name.
The communication with the server use the standard port 25. A different port can be specified in the Server_Add parameter adding the port number (after the separator character ':').
The SSL connections require an enabling and configuration using the file CHIP.INI (refer to the administration pages).
The E-mail address of the recipient is set into To_Address parameter.
The Subject parameter is a string that describe the object of the E-mail.
Any attached file to the E-mail can be indicated by the File parameter. The attached file must reside on a accessible disk of the system.
The text of E-mail must be passed using the Text string. The use special of characters (as $N for new line and $L for line feed) are allowed.

Example

```
Result := SendMail('192.168.1.101',
                   'MyTarget@gmx.com',
                   'MyTarget',
                   '213.165.64.45',
                   'username',
                   'password',
                   'info@destination.com',
                   'E-mail test',
                   'A:\PLC_PRG\Attached.txt',
                   'This is the E-mail text'
                   );
```

1.12.4 SNTP_DateTime

FUNCTION SNTP_DateTime : BOOL

VAR_INPUT
  Server_IP : *STRING*;
  TimeZone : *INT*;
  Daylight_Save : *BYTE*;
END_VAR

Synchronizes Real Time Clock with the specified NTP server.

Return Value

TRUE : success
FALSE : error

Input Parameters

Server_IP

IP address of NTP server

TimeZone

Geographical zone offset (UTC+n)

Daylight_Save

Daylight saving enable (+1h)

Comments

This function connects to a NTP server and request the reference date and time.
The IP address of NTP server can be specifies in the format xxx.xxx.xxx.xxx or with the string of address name (in this case the internal DNS resolver is used).
TimeZone parameter identifies the geographical zone (for example +1 for Rome). This offset is added to the UTC (Coordinated Universal Time) returned by NTP server.
Daylight_Save enable the sum of one hour to take account of daylight saving. This enabling is diversified for every country so its management request the knowledge of the specific situation. With the value 0 of the parameter does not add one hour (winter), while with the value 1 is added one hour (summer). The values above 1 encoding zones for the automatic calculation of daylight saving. Currently it is implemented the management of value 2 relative to Europe.
The resultant value of date and time is finally forced into the Real Time Clock device of PLC.

NOTE: by enabling the automatic NTP synchronization with the appropriate configuration parameters of CHIP.INI file (see NTP section of the configuration Web interface), a single NTP synchronization is executed every time the PLC power supply is started. Successive NTP synchronizations with CHIP.INI default parameters can be then triggered inside IEC program calling the function with null input values: SNTP_DateTime('', 0, 0).

Example

```
Result := SNTP_DateTime('62.206.250.163', +1, TRUE);
```

1.13 The MySQL library (MySQL_Lib.lib)

This library offers the functions for the management of the MySQL databases.
MySQL is one of the world's most used database and its data access is based on SQL (Structured Query Language) protocol. A detailed documentation can be found at the following link http://www.mysql.com.
Most of Web hosting provider offers a service for the MySQL database. In this way, you can access directly to the database using various tools provided in the control panel of the service.
There are also several tools program, running on your PC, to access and manage a remote MySQL database. One of these is http://www.heidisql.com which is a powerful and free tool to create and interact with a local or remote database.
In addition there are various tools for the direct installation of a MySQL database on your server or PC as href="http://www.apachefriends.org/en/xampp.html or http://www.wampserver.com/en/download.php. These tools, installed on a local server, allow both the running of PHP scripts and the MySQL database management, as it happens with the external Web hosting services.
The MySQL library consists of CoDeSys functions that access, using the HTTP client, to a PHP server for the execution of a script as interface with MySQL. For this you need to pre-install the MySQL folder containing the PHP code on a server that can also be the same that runs the MySQL database. The PHP code, triggered by the library functions, performs a specific queries on the database and returns data in response. Later, other library functions can access to the data received by the query.

Here is a list of all the functions this library offers:

MySQL_Connect_Set
MySQL_Data_Seek
MySQL_Database_Set
MySQL_Fetch_Row
MySQL_Field_Len
MySQL_Field_Name
MySQL_Num_Fields
MySQL_Num_Rows
MySQL_Query
MySQL_Result

1.13.1 MySQL_Connect_Set

FUNCTION MySQL_Connect_Set : BOOL

```
VAR_INPUT
   Php_Url : STRING;
   Server : STRING;
   Username : STRING;
   Password : STRING;
END_VAR
```

Sets the parameters for connecting to MySQL database.

Return Value

Always TRUE

Input Parameters

Php_Url

URL path containing the PHP interface page

Server

Address of the server for MySQL database access

Username

Username for database access

Password

Password for database access

Comments

This function sets the connection parameters to be used for all subsequent operations on the MySQL database.
The database management is done through the HTTP client access to the PHP page as interface with the MySQL server. The mysql_query.php file must be installed on a PHP server, set by the Php_Url parameter, not necessarily the same server used for MySQL database.
The Server parameter sets the address of the database host and is usually "localhost" if the same server running the PHP interface code.
The Username and Password parameters must match the authorization for the user enabled to perform operations on the MySQL database.

Related Topics

MySQL_Database_Set

1.13.2 MySQL_Data_Seek

FUNCTION MySQL_Data_Seek : BOOL

```
VAR_INPUT
   Row : UINT;
END_VAR
```

Forces the row pointer for result set reading.

Return Value

TRUE : success
FALSE : error

Input Parameters

Row

Position of the next line read from the result set

Comments

The function sets the pointer position of the next row of result set to be read with the MySQL_Fetch_Row function.
After a query, the pointer is positioned to value 0 corresponding to read the first row of the result set. Each sequential row read increment the pointer by one position. With this function you can reposition the cursor on a row between 0 and Num_Rows-1 where Num_Rows is the value returned by the MySQL_Num_Rows.
The function returns 1 if successful and returns 0 if the location requested is not available.

Related Topics

MySQL_Fetch_Row

1.13.3 MySQL_Database_Set

FUNCTION MySQL_Database_Set : BOOL

```
VAR_INPUT
   Database : STRING;
END_VAR
```

Selects the MySQL database currently in use.

Return Value

Always TRUE

Input Parameters

Database

Name of the current database to use

Comments

This function sets the name of the MySQL database to be used for all subsequent management functions.
The management is done by running the PHP script that initially make the MySQL connection using the parameters set by the MySQL_Connect_Set function. Then the database selection is made using the name set by the MySQL_Database_Set function. After running the script interface and returning the response, the database connection is closed.

Related Topics

MySQL_Connect_Set

1.13.4 MySQL_Fetch_Row

FUNCTION MySQL_Fetch_Row : STRING

```
VAR_INPUT
   Row : STRING;
END_VAR
```

Extracts a row from the query result set.

Return Value

The same string pointer passed as parameter

Input Parameters

Row

Pointer of string where to copy the row readed

Comments

The database queries such as SELECT, SHOW, DESCRIBE, EXPLAIN returns a result set consisting of whole rows (table records) that satisfy the condition of the query. The query function saves the entire result set into a memory buffer from where you can sequentially extract individual rows by MySQL_Fetch_Row function calls.
Each fetched row is copied into the string pointed by the Row parameter passed to the function and contains all the columns (fields) of the record separated by semicolons (plus space). All fields of string type are delimited by single quotes (apostrophes).
If the row is not available returns the empty string.

Related Topics

MySQL_Data_Seek

1.13.5 MySQL_Field_Len

FUNCTION MySQL_Field_Len : UINT

```
VAR_INPUT
   Field : UINT;
END_VAR
```

Reads the length of a field of the database table.

Return Value

Length of the field in the table

Input Parameters

Field

Number of field for which read the length

Comments

The function allows to know the length of a specific field of the table following the execution of a query that returns a result set. The length is determined at creation of the field in the table and also defines the memory used to store values.

Related Topics

MySQL_Num_Fields

MySQL_Field_Name

1.13.6 MySQL_Field_Name

FUNCTION MySQL_Field_Name : STRING

```
VAR_INPUT
   Name : STRING;
   Field : UINT;
END_VAR
```

Reads the name of a database table field.

Return Value

The same string pointer passed as parameter

Input Parameters

Name

Pointer of string where to copy the name readed

Field

Number of field for which read the name

Comments

This feature allows you to read the name associated with one of the fields (columns) of the table following the execution of a query that returns a result set.
The parameter Field indicates the position of the column between 0 and Num_Fields-1 where Num_Fields is the value returned by MySQL_Num_Fields.
The field name is copied to the string pointed by the Name parameter passed to the function.
If the name is not available returns the empty string.

Related Topics

MySQL_Num_Fields

MySQL_Field_Len

1.13.7 MySQL_Num_Fields

FUNCTION MySQL_Num_Fields : UINT

Reads the current number of fields after a query execution.

Return Value

Number of fields in the table related to the query

Input Parameters

None

Comments

This function allows you to read the number of fields (columns) available in the table following the execution of a query that returns a result set.

Related Topics

MySQL_Num_Rows

1.13.8 MySQL_Num_Rows

FUNCTION MySQL_Num_Rows : UINT

Reads the current number of rows after a query execution.

Return Value

Number of rows in the table related to the query

Input Parameters

None

Comments

This feature allows you to read the number of rows in the table after a query execution.
For queries that return a result set the readed number of rows corresponds to many records in the table have been affected by the query. For example, after a "DELETE FROM table WHERE condition" returns the number of deleted rows in the table according to the WHERE condition.
For query functions that return a result set, the number of rows resulting from the function corresponds to the number of records readed from the table according to a certain condition and that can be readed with subsequent MySQL_Fetch_Row function calls.

Related Topics

MySQL_Num_Fields

1.13.9 MySQL_Query

FUNCTION MySQL_Query : STRING

```
VAR_INPUT
   Query : STRING;
END_VAR
```

Sends a query to the MySQL database.

Return Value

0 : success
-1...-4: DNS client error
201...265: TCP/IP functions error
1/2/4/8/16: SSL functions error
1001: Server parameter is not defined
1002: Username parameter is not defined
1003: Password parameter is not defined
1004: Database parameter is not defined
1005: Query parameter is not defined
2001: MySQL connection error
2002: Error selecting database
2003: Error executing query
3001: no tag received for the error code
3002: no tag received for number of rows (records)
3003: no tag received for result set available
3004: no tag received for number of fields (columns)
3005: no tag received for field names
3006: no tag received for field lengths
3007: no tag received for the result set row

Input Parameters

Query

The query string to execute on the MySQL database

Comments

This function sends a query to the MySQL database that is currently selected by MySQL_Database_Set.
To prepare the query string, passed directly to the database server, refer to the official MySQL documentation.
The query INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, do not return a result set (a set of data readed from the database).
Queries such as SELECT, SHOW, DESCRIBE, EXPLAIN instead save the result set into a buffer from which are then extracted values of the table rows with appropriate fetch functions.
The function returns 0 if it executes without error, otherwise returns the error code. The error code can be any of those provided by the function HTTP_Get used to access the interface PHP page, or a specific error code of PHP script execution.

Related Topics

MySQL_Connect_Set

MySQL_Database_Set

1.13.10 MySQL_Result

FUNCTION MySQL_Result : STRING

```
VAR_INPUT
   Cell : STRING;
   Row : UINT;
   Field : UINT;
END_VAR
```

Extracts a single cell of the query result set.

Return Value

The same string pointer passed as parameter

Input Parameters

Cell

Pointer of string where to copy the cell readed

Row

Row number of the cell to read

Field

Field number of the cell to read

Comments

The database queries such as SELECT, SHOW, DESCRIBE, EXPLAIN returns a result set consisting of whole rows (table records) that satisfy the condition of the query. The query function saves the entire result set into a memory buffer from which can then extract the value of a single cell with MySQL_Result function, directly specifying the line and column number using the Row and Field parameters.
The value of the cell is returned as a string to the position defined by the pointer passed as a Cell parameter. Furthermore, if the contents of the cell is a variable of string type, the result is delimited by single quotes (apostrophe) as it does for MySQL_Fetch_Row function.
This function accesses randomly, and not sequentially as MySQL_Fetch_Row, to a single cell of the table (a cross between the row and column of the result set). However, to read more cells in a row you should use the MySQL_Fetch_Row that is faster than MySQL_Result.
If the cell is not available returns the empty string.

Related Topics

MySQL_Num_Rows

MySQL_Num_Fields

1.14 The RS485/Serial library (rs485.lib)

This library offers extended functionality for serial communication via the RS485 protocol. Especially for controlling an RS485 transceiver, because here the transmitter may only be active while data is actually sent.
The available port numbers are enumerated in RS485_PORTS.

This library offers the following functions and data types:

TYPE RS485_BREAK
TYPE RS485_FLOWCTRL
TYPE RS485_MODE
TYPE RS485_PARITY
TYPE RS485_PORTS
Rs485ComClose
Rs485ComOpen
Rs485FlushOutput
Rs485GetStatus
Rs485IsByteAvailable
Rs485PurgeInput
Rs485PurgeOutput
Rs485ReceiveBlock
Rs485ReceiveByte
Rs485SendBlock
Rs485SendBreak
Rs485SendByte
Rs485SetFlowcontrol
Rs485SetMode

1.14.1 TYPE RS485_BREAK

TYPE RS485_BREAK

```
TYPE RS485_BREAK :
(
    RS485_BREAK_LONG := 1,
    RS485_BREAK_SHORT := 2,
    RS485_BREAK_EXTRALONG := 3
);
END_TYPE
```

This type enumerates the desired length of a break pulse.

Member

RS485_BREAK_LONG := 1

long break (Duration: > 1 character)

RS485_BREAK_SHORT := 2

short break (Duration: > 2 characters)

RS485_BREAK_EXTRALONG := 3

extra long break (Duration: > 3 characters)

1.14.2 TYPE RS485_FLOWCTRL

TYPE RS485_FLOWCTRL

```
TYPE RS485_FLOWCTRL :
(
    RS485_FLOWCTRL_OFF := 0,
    RS485_FLOWCTRL_XONXOFF_SEND := 1,
    RS485_FLOWCTRL_RTSCTS := 2,
    RS485_FLOWCTRL_XONXOFF_RECV := 8,
    RS485_FLOWCTRL_XONXOFF_SEND_RECV := 9
);
END_TYPE
```

This type enumerates the desired method of flow control.

Member

RS485_FLOWCTRL_OFF := 0

no flow control

RS485_FLOWCTRL_XONXOFF_SEND := 1

XONXOFF on transmit (watch for XOFF while sending)

RS485_FLOWCTRL_RTSCTS := 2

CTS on transmit, RTS on receive

RS485_FLOWCTRL_XONXOFF_RECV := 8

XONXOFF on receive (send XOFF when buffer nearly full)

RS485_FLOWCTRL_XONXOFF_SEND_RECV := 9

XONXOFF used on both Tx and Rx

1.14.3 TYPE RS485_MODE

TYPE RS485_MODE

```
TYPE RS485_MODE :
(
    RS485_MODE_LOWACTIVE := 0,
    RS485_MODE_HIGHACTIVE := 1,
    RS485_DISABLE := 2
);
END_TYPE
```

This type enumerates the desired method.

Member

RS485_MODE_LOWACTIVE := 0

Tx Enable low active

RS485_MODE_HIGHACTIVE := 1

Tx Enable high active

RS485_DISABLE := 2

Disable RS485 Mode

NOTE: normally this parameter must be set to RS485_MODE_HIGHACTIVE.

1.14.4 TYPE RS485_PARITY

TYPE RS485_PARITY

```
TYPE RS485_PARITY :
(
    RS485_PARITY_NO := 0,
    RS485_PARITY_ODD := 1,
    RS485_PARITY_EVEN := 2,
    RS485_PARITY_MARK := 3,
    RS485_PARITY_SPACE := 4
);
END_TYPE
```

This type enumerates the desired parity setting of a serial port.

Member

RS485_PARITY_NO := 0

no parity

RS485_PARITY_ODD := 1

odd parity

RS485_PARITY_EVEN := 2

even parity

RS485_PARITY_MARK := 3

mark parity

RS485_PARITY_SPACE := 4

space parity

1.14.5 TYPE RS485_PORTS

TYPE RS485_PORTS

```
TYPE RS485_PORTS :
(
    RS485_COM1 := 1,
    RS485_COM2 := 2,
    RS485_COM3 := 3,
    RS485_COM4 := 4,
    RS485_COM5 := 5,
    RS485_COM6 := 6,
    RS485_COM7 := 7,
    RS485_COM8 := 8
);
END_TYPE
```

This type enumerates the desired serial port.

Member

RS485_COM1 := 1

Uart0

RS485_COM2 := 2

Uart1

RS485_COM3 := 3

Uart2

RS485_COM4 := 4

Uart3

RS485_COM5 := 5

Virtual Serial Interface via USB

RS485_COM6 := 6

Virtual Serial Interface via USB

RS485_COM7 := 7

Virtual Serial Interface via USB

RS485_COM8 := 8

Virtual Serial Interface via USB

1.14.6 Rs485ComClose

FUNCTION Rs485ComClose : **BOOL**

```
VAR_INPUT
   handle : DWORD;
END_VAR
```

Closes a serial communications port.

Return Value

TRUE : success
FALSE : error

Input Parameters

handle

Port handle returned by Rs485ComOpen

Comments

Use this function to close a port that is no longer used.

Related Topics

Rs485ComOpen

1.14.7 Rs485ComOpen

FUNCTION Rs485ComOpen : **DWORD**

VAR_INPUT
  port : ***RS485_PORTS***;
  baud : ***DINT***;
  parity : ***RS485_PARITY***;
  wordlen : ***INT***;
  stopbits : ***INT***;
END_VAR

Opens a serial port.

Return Value

Returns 16#FFFFFFFF when called with invalid parameters.
Otherwise a port handle is returned. This handle has to be used in further calls to this interface.

Input Parameters

port

Number of desired port (enumeration type)

baud

Baud rate (110..115200)

parity

Parity (enumeration type)

wordlen

Data length (7 or 8 bits)

stopbits

Number of stop bits (1 or 2)

Related Topics

Rs485ComClose

Rs485SetFlowcontrol

1.14.8 Rs485FlushOutput

FUNCTION Rs485FlushOutput : **BOOL**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Waits until all data from the output buffer has been sent.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

Comments

Be careful when using this function. The controller will wait until the function is finished. This might take a while if the baud rate is low and there are many characters in the buffer to send.

Related Topics

Rs485PurgeOutput

Rs485PurgeInput

1.14.9 Rs485GetStatus

FUNCTION Rs485GetStatus : **INT**

VAR_INPUT
  handle : *DWORD*;
END_VAR

Returns the status of a serial port.

Return Value

Returns a status code (bit field)

Input Parameters

handle

Port handle returned by Rs485ComOpen

Comments

The status bits have the following meaning:

```
bit 6: RS485_OUTPUT_BUFFER_EMPTY (tx buffer is empty)
bit 5: RS485_OUTPUT_NOT_FULL     (tx buffer is not full)
bit 4: RS485_LINE_BREAK          (line break detected)
bit 3: RS485_FRAMING_ERROR       (framing error detected)
bit 2: RS485_PARITY_ERROR        (parity error detected)
bit 1: RS485_OVERRUN_ERROR       (overrun occurred on receiver)
bit 0: RS485_DATA_AVAILABLE      (data is available in rx buffer)
```

Related Topics

Rs485SetFlowcontrol

Rs485SendBreak

Rs485ComOpen

1.14.10 Rs485IsByteAvailable

FUNCTION Rs485IsByteAvailable : **INT**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Peek if received data is available, without removing the data from the internal receive buffer.

Return Value

Returns the received byte or -1 in case of an invalid parameter or no data available

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

Comments

The only difference to Rs485ReceiveByte is that with Rs485IsByteAvailable the received byte is not removed from the receive buffer.

Related Topics

Rs485ReceiveByte

1.14.11 Rs485PurgeInput

FUNCTION Rs485PurgeInput : **BOOL**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Discards the contents of the serial receive buffer.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

Comments

All data in the serial receive buffer will be discarded and not be read.

Related Topics

Rs485FlushOutput

Rs485PurgeOutput

1.14.12 Rs485PurgeOutput

FUNCTION Rs485PurgeOutput : **BOOL**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Discards the contents of the serial transmit buffer.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

Comments

All data in the serial transmit buffer will be discarded and not be sent.

Related Topics

Rs485FlushOutput

Rs485PurgeInput

1.14.13 Rs485ReceiveBlock

FUNCTION Rs485ReceiveBlock : **UINT**

```
VAR_INPUT
   dwHandle : DWORD;
   pBuffer : POINTER to BYTE;
   count : UINT;
END_VAR
```

Receives a block of data from a serial port. Returns immediately if no data has been received.

Return Value

Returns the number of bytes transferred to the buffer

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

pBuffer

Pointer to buffer where to store received data

count

Maximum number of bytes to put into the buffer

Related Topics

Rs485ReceiveByte

Rs485SendBlock

Rs485SendByte

1.14.14 Rs485ReceiveByte

FUNCTION Rs485ReceiveByte : **INT**

```
VAR_INPUT
   dwHandle : DWORD;
END_VAR
```

Receives a byte of data from a serial port. Returns immediately if no data has been received.

Return Value

Returns the received byte or -1 in case of an invalid parameter or no data available

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

Related Topics

Rs485SendByte

Rs485SendBlock

Rs485ReceiveBlock

1.14.15 Rs485SendBlock

FUNCTION Rs485SendBlock : **UINT**

```
VAR_INPUT
   dwHandle : DWORD;
   pBuffer : POINTER to BYTE;
   count : UINT;
END_VAR
```

Sends a block of data via a serial port.

Return Value

Returns the number of bytes sent

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

pBuffer

Pointer to data to send (can be obtained using the ADR() function)

count

Number of bytes to send

Related Topics

Rs485SendByte

Rs485ReceiveByte

Rs485ReceiveByte

1.14.16 Rs485SendBreak

FUNCTION Rs485SendBreak : **BOOL**

VAR_INPUT
  dwHandle : ***DWORD***;
  length : ***RS485_BREAK***;
END_VAR

Sends a break signal on a serial port.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

length

Length of the break pulse (enumeration type)

Comments

A short break is a continuous Low on the TXD output for a duration of more than one frame transmission time M, where: `M = startbit + data bits (+ parity bit) + stop bit.`
A long break is a continuous Low on the TXD output for a duration of more than two frame transmission times plus the transmission time for three additional bits (2M+3).
An extra long break is a continuous Low on the TXD output for a duration of more than three frame transmission times.

Related Topics

Rs485GetStatus

1.14.17 Rs485SendByte

FUNCTION Rs485SendByte : **INT**

```
VAR_INPUT
   dwHandle : DWORD;
   ch : BYTE;
END_VAR
```

Sends a byte of data to a serial port.

Return Value

Returns 0 if no buffer space to store character available, or 1 on success, or -1 on invalid parameter

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

ch

Character to send

Related Topics

Rs485ReceiveByte

Rs485SendBlock

Rs485ReceiveBlock

1.14.18 Rs485SetFlowcontrol

FUNCTION Rs485SetFlowcontrol : **USINT**

VAR_INPUT
  dwHandle : ***DWORD***;
  flowctrl : ***RS485_FLOWCTRL***;
END_VAR

Sets the flowcontrol method of a serial port.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

flowctrl

Flowcontrol method (bit field / enumeration type)

Comments

The RS485_FLOWCTRL enumeration type is also declared in this library.

Related Topics

Rs485ComOpen

1.14.19 Rs485SetMode

FUNCTION Rs485SetMode : **BOOL**

VAR_INPUT
  dwHandle : ***DWORD***;
  mode : ***RS485_MODE***;
END_VAR

Selects the operating mode of the port.

Return Value

TRUE : success
FALSE : error

Input Parameters

dwHandle

Port handle returned by Rs485ComOpen

mode

Mode (enumeration type)

Comments

This function should be the first function called before any data exchange can take place.

Related Topics

Rs485ComOpen

1.15 The RS485/Expansion library (RS485_Lib.lib)

This library offers the functions for the management of the RS485 slave expansion.
A complete refresh of slave expansion resources is obtained using this functions into IEC program.

Here is a list of all the functions and data type this library offers:

DMX512_Tx
RS485_Init_A
RS485_Tx_Rx_A

1.15.1 DMX512_Tx

FUNCTION DMX512_Tx : UINT

VAR_INPUT
  Port : *PORTS*;
  Offset : *UINT*;
  Data_Ptr : *DWORD*;
  Data_Num : *UINT*;
END_VAR

Executes the transmission of a DMX512 frame.

Return Value

0 : success
200 : bad parameter
201 : serial initialization error

Input Parameters

Port

Serial port number

Offset

Position of Data from the start of DMX512 frame

Data_Ptr

Pointer of Data buffer to send

Data_Num

Number of Data bytes

Comments

This function sends over RS485 port a sequence of bytes in accordance with the DMX512 protocol. The speed of communication is 250kbit/s and for synchronization is used a break of 3 characters (132us) followed by a start code (one byte of 0 value). Then the buffer data bytes are sent in quantities up to 512.
The data is transmitted with a start bit, 8 data bits and 2 stop bits. The Offset parameter determines a transmission of data to 0, in an amount equal to Offset-1, before sending the bytes of the data buffer. Normally this parameter is 1, since the data to be transmitted are justified by the DMX512 address 1.
Data_Ptr parameter is the pointer to the first byte of data buffer to be transmitted.
Data_Num parameter is the number of bytes in the buffer to be transmitted.

1.15.2 RS485_Init_A

FUNCTION RS485_Init_A : BOOL

VAR_INPUT
  Port : *PORTS*;
  Baudrate : *UDINT*;
  Timeout : *UINT*;
  Tx_Delay : *UINT*;
END_VAR

Initializes the RS485 type A expansion communication.

Return Value

Always TRUE

Input Parameters

Port

Serial port number (default to COM2=2)

Baudrate

Speed of communication (default to 115200 bits/s)

Timeout

Maximum wait for data reception (default to 2ms)

Tx_Delay

Bytes spacing during transmission (default to 0)

Comments

Use this function to initialize the RS485 type A communication with expansion slaves.
This function is not necessary in the case of communications with default parameter values.

Related Topics

RS485_Tx_Rx_A

1.15.3 RS485_Tx_Rx_A

FUNCTION RS485_Tx_Rx_A : UINT

VAR_INPUT
  Add_Cmd : *BYTE*;
  Tx_Num : *UINT*;
  Rx_Num : *UINT*;
  IO_Ptr : *DWORD*;
  Swap : *USINT*;
END_VAR

Executes the RS485 type A expansion communication.

Return Value

0 : success
200 : bad parameter
201 : serial initialization error
202 : slave has received a bad checksum
203 : master has received a bad checksum
204 : timeout during master reception

Input Parameters

Add_Cmd

Header byte with formatted slave address and command

Tx_Num

Number of bytes for output area

Rx_Num

Number of bytes for input area

IO_Ptr

Pointer to output/input area buffer

Swap

Swap bytes of word (2) or dword (4)

Comments

This function executes a complete sequence of RS485 type A communication with expansion slaves.
Type A communication works as follows.
The value of Add_Cmd parameter, that contains the address/command, is transmitted with the 9° bit forced to 1. If TxNum > 0 are then transmitted TxNum bytes pointed by IO_Ptr. The checksum (XOR of the only data bytes) is finally trasmitted. At the end is attended the 0 value to confirm the correct reception by the slave.
Else if Rx_Num > 0 is attended the reception of Rx_Num bytes plus checksum, saving the values in a temporary buffer. Verified the checksum, the buffer is copied to the area pointed by IO_Ptr.
The Swap parameter flips the order of the bytes (from big endian to little endian and vice versa): for word values set Swap to 2 and for dword values set Swap to 4. This is also applied for array of words or dwords.

Related Topics

RS485_Init_A

1.16 The MODBUS library (MODBUS_Lib.lib)

This library offers the functions for the management of the slave expansions using MODBUS protocol.
A complete refresh of slave expansion resources is obtained using this functions into IEC program.

Here is a list of all the functions this library offers:

TYPE MB_RTU_PARITY
TYPE MODBUS_STATUS
MB_RTU_Deinit
MB_RTU_Init
MB_RTU_Rd_Coils
MB_RTU_Rd_Hold_Regs
MB_RTU_Rd_Input_Regs
MB_RTU_Rd_Inputs
MB_RTU_Req_Rsp
MB_RTU_Wr_Coils
MB_RTU_Wr_Hold_Regs
MB_RTU_Wr_Rd_Hold_Regs
MB_RTU_Wr_Single_Coil
MB_RTU_Wr_Single_Reg
MB_RTU_Slave
MB_TCP_Connect
MB_TCP_Disconnect
MB_TCP_Rd_Coils
MB_TCP_Rd_Hold_Regs
MB_TCP_Rd_Input_Regs
MB_TCP_Rd_Inputs
MB_TCP_Req_Rsp
MB_TCP_Wr_Coils
MB_TCP_Wr_Hold_Regs
MB_TCP_Wr_Rd_Hold_Regs
MB_TCP_Wr_Single_Coil
MB_TCP_Wr_Single_Reg
MB_TCP_Server
MB_Swap_Dword
MB_Swap_Dword2
MB_Swap_Word

1.16.1 MB_RTU_PARITY

TYPE MB_RTU_PARITY

```
TYPE MB_RTU_PARITY :
(
    PARITY_NO := 0,
    PARITY_ODD := 1,
    PARITY_EVEN := 2,
    PARITY_NO_1S := 3
);
END_TYPE
```

This type enumerates the desired parity setting of a serial port.

Member

PARITY_NO := 0

no parity, 2 stop bits

PARITY_ODD := 1

odd parity, 1 stop bit

PARITY_EVEN := 2

even parity, 1 stop bit

PARITY_NO_1S := 3

no parity, 1 stop bit

NOTE: In the case of "No parity" the protocol specification requires the presence of two stop bits. However, for compatibility with other systems that do not comply with this standard, is also available "No parity" mode with only one stop bit. Below is an extract of the official specific of MODBUS RTU protocol:

2.5.1 RTU Transmission Mode

When devices communicate on a MODBUS serial line using the RTU (Remote Terminal Unit) mode, each 8–bit byte in a message contains two 4–bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII mode for the same baud rate. Each message must be transmitted in a continuous stream of characters.

The format (11 bits) for each byte in RTU mode is :

Coding System: 8–bit binary
Bits per Byte: 1 start bit
8 data bits, least significant bit sent first
1 bit for parity completion
1 stop bit

Even parity is required, other modes (odd parity, no parity) may also be used. In order to ensure a maximum compatibility with other products, it is recommended to support also No parity mode. The default parity mode must be even parity.

Remark : the use of no parity requires 2 stop bits.

How Characters are Transmitted Serially :

Each character or byte is sent in this order (left to right):

Least Significant Bit (LSB) . . . Most Significant Bit (MSB)

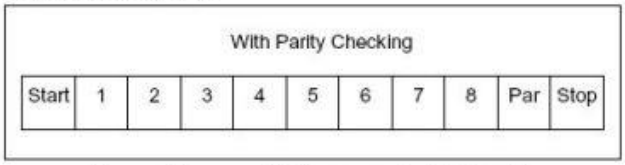

Figure 10: Bit Sequence in RTU mode

Devices may accept by configuration either Even, Odd, or No Parity checking. If No Parity is implemented, an additional stop bit is transmitted to fill out the character frame to a full 11-bit asynchronous character :

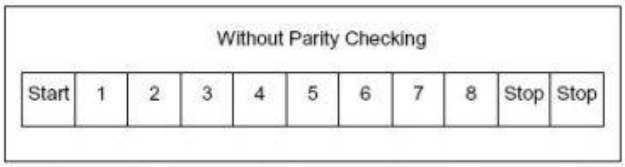

Figure 11: Bit Sequence in RTU mode (specific case of No Parity)

Frame Checking Field : Cyclical Redundancy Checking (CRC)

1.16.2 MODBUS_STATUS

TYPE MODBUS_STATUS

```
TYPE MODBUS_STATUS :
(
    MODBUS_STANDBY := 0,
    MODBUS_ACTIVE := 100,
    MODBUS_ERROR := 200,
    MODBUS_PARAMETER_ERR := 201,
    MODBUS_OPEN_ERR := 202,
    MODBUS_OPTION_ERR := 203,
    MODBUS_BIND_ERR := 204,
    MODBUS_LISTEN_ERR := 205
);
END_TYPE
```

This type enumerates the current status of MODBUS slave/server service.

Member

MODBUS_STANDBY := 0

Service not activated

MODBUS_ACTIVE := 100

Service activated

MODBUS_ERROR := 200

Service stopped due to error

MODBUS_PARAMETER_ERR := 201

Service stopped: parameter not valid

MODBUS_OPEN_ERR := 202

Service stopped: error opening port (RTU) / socket (TCP)

MODBUS_OPTION_ERR := 203

Service stopped: error setting socket option

MODBUS_BIND_ERR := 204

Service stopped: error binding listen socket to local address

MODBUS_LISTEN_ERR := 205

Service stopped: error activating listen on socket

Related Topics

MB_RTU_Slave

MB_TCP_Server

1.16.3 MB_RTU_Deinit

FUNCTION MB_RTU_Deinit : BOOL

Deinitializes the MODBUS RTU on RS485/RS232 for expansion communication.

Return Value

Always TRUE

Input Parameters

None

Comments

Use this function to deinitialize the MODBUS RTU communication over RS485/RS232 with expansion slaves.
Normally for the continuous refreshing of the slave resources, the communication is initialized only at poweron, leaving always on. So this function is not normally used.

Related Topics

MB_RTU_Init

1.16.4 MB_RTU_Init

FUNCTION MB_RTU_Init : UINT

VAR_INPUT
  Port : *PORTS*;
  Baudrate : *UDINT*;
  Parity : *MB_RTU_PARITY*;
  Rsp_Timeout : *UINT*;
END_VAR

Initializes the MODBUS RTU on RS485/RS232 for expansion communication.

Return Value

0 : success
128 : initialization parameter not valid
129 : inizialization error

Input Parameters

Port

Serial port number

Baudrate

Speed of communication (300-1M bits/s)

Parity

Parity select (0/1/2=NO/ODD/EVEN)

Rsp_Timeout

Maximum wait (ms) for slave response reception (for example 10)

Comments

Use this function to initialize the MODBUS RTU communication over RS485/RS232 with expansion slaves.
This function must be called before executing a MODBUS communication. Normally for the continuous refreshing of the slave resources, the communication is initialized only at poweron, leaving always on.

Related Topics

MB_RTU_Deinit

1.16.5 MB_RTU_Rd_Coils

FUNCTION MB_RTU_Rd_Coils : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS RTU function code 1: Read Coils.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of first bit to be read

Quantity

Number of bits to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more bits of memory (Coils) from a specific slave using the MODBUS RTU protocol.
This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the starting bit address to read.
Quantity parameter indicates the number of bits to be read.
Data parameter is the pointer of the buffer in which receive the values of the bits. The LSB of the first byte of the buffer contains the addressed bit. The amount of bytes placed in the receive buffer is equal to: Data_Len = Quantity / 8, where if remainder > 0 then Data_Len = Data_Len + 1.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Wr_Coils

MB_RTU_Wr_Single_Coil

1.16.6 MB_RTU_Rd_Hold_Regs

FUNCTION MB_RTU_Rd_Hold_Regs : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS RTU function code 3: Read Holding Registers.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of first word to be read

Quantity

Number of words to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more words of memory (Holding Registers) from a specific slave using the MODBUS RTU protocol.
This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the starting word address to read.
Quantity parameter indicates the number of words to be read.
Data parameter is the pointer of the buffer in which receive the values of the words. The amount of bytes placed in the receive buffer is equal to: Data_Len = 2 * Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Wr_Hold_Regs

MB_RTU_Wr_Single_Reg

1.16.7 MB_RTU_Rd_Input_Regs

FUNCTION MB_RTU_Rd_Input_Regs : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS RTU function code 4: Read Input Registers.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of first word to be read

Quantity

Number of words to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more words of input (Input Registers) from a specific slave using the MODBUS RTU protocol. This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the starting word address to read.
Quantity parameter indicates the number of words to be read.
Data parameter is the pointer of the buffer in which receive the values of the words. The amount of bytes placed in the receive buffer is equal to: Data_Len = 2 * Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Rd_Inputs

1.16.8 MB_RTU_Rd_Inputs

FUNCTION MB_RTU_Rd_Inputs : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS RTU function code 2: Read Discrete Inputs.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of first bit to be read

Quantity

Number of bits to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more bits of input (Discrete Inputs) from a specific slave using the MODBUS RTU protocol. This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the starting bit address to read.
Quantity parameter indicates the number of bits to be read.
Data parameter is the pointer of the buffer in which receive the values of the bits. The LSB of the first byte of the buffer contains the addressed bit. The amount of bytes placed in the receive buffer is equal to: Data_Len = Quantity / 8, where if remainder > 0 then Data_Len = Data_Len + 1.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Rd_Input_Regs

1.16.9 MB_RTU_Req_Rsp

FUNCTION MB_RTU_Req_Rsp : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Fun_Code : *BYTE*;
  Req_Data : *DWORD*;
  Req_Len : *WORD*;
  Rsp_Data : *DWORD*;
  Rsp_Len : *WORD*;
END_VAR

Executes a complete MODBUS RTU request tx and response rx on RS485/RS232 for expansion communication.

Return Value

0 : success
1 : slave has received a not valid function code
2 : slave has received a not valid object address
3 : slave has received a not valid object data
4 : non-recoverable error during execution of the slave function
5 : response to prevent a master receive timeout while slave performs a long operation
6 : slave is busy in the execution of required function
130 : timeout during master reception
131 : received from slave a bad parity
132 : received from slave a bad checksum
133 : bad slave address in replay
134 : bad function code in replay
135 : bad number of bytes transmitted
136 : bad number of bytes received

Input Parameters

Slave_Add

Slave address (valid 1-247)

Fun_Code

MODBUS function code

Req_Data

Pointer to request data buffer

Req_Len

Number of request data bytes

Rsp_Data

Pointer to response data buffer

Rsp_Len

Number of response data bytes aspected

Comments

This function execute a complete MODBUS RTU communication sequence over RS485/RS232 port with expansion slaves.
Slave_Add parameter select the specific slave on the serial bus.
Fun_Code parameter is the code of the function to execute.
Req_Data parameter is the pointer to the buffer with the data to be transmitted for the specific function code.
Req_Len parameter indicates the number of bytes in the Req_Data buffer to be transmitted to the slave.
Rsp_Data parameter is the pointer to the buffer in which receive the data of response to the function from the slave.
The Rsp_Len parameter indicates the number of data bytes expected in the Rsp_Data receive buffer as response to the specific function code.

The function returns 0 if performed without error, otherwise it returns the error code.
To use the MODBUS communication function refer to the technical documentation "MODBUS Application Protocol Specification" downloadable at http://www.modbus-ida.org./tech.php.

In the following example one input word and one output word are refreshed for the slave at address 1:

```
PROGRAM MB_RTU
VAR
    MB_RTU_Initialized: BOOL := FALSE;
    Req_Buf: ARRAY[0..10] OF BYTE;
    Rsp_Buf: ARRAY[0..10] OF BYTE;
    Input_Reg: WORD;
    Output_Reg: WORD;
END_VAR

(* Initialize MODBUS communication *)
IF NOT MB_RTU_Initialized THEN
    MB_RTU_Init(COM2, 19200, PARITY_EVEN, 100);
    MB_RTU_Initialized := TRUE;
END_IF

(* Read one input word at address 0 from Slave 1 *)
Req_Buf[0] := 0;
Req_Buf[1] := 0;
Req_Buf[2] := 0;
Req_Buf[3] := 1;
IF MB_RTU_Req_Rsp(1, 4, ADR(Req_Buf), 4, ADR(Rsp_Buf), 3) = 0 THEN
    Input_Reg := 256 * Rsp_Buf[1] + Rsp_Buf[2];
END_IF

(* Write one output word at address 1 to Slave 1 *)
Req_Buf[0] := 0;
Req_Buf[1] := 1;
Req_Buf[2] := Output_Reg / 256;
Req_Buf[3] := Output_Reg MOD 256;
MB_RTU_Req_Rsp(1, 6, ADR(Req_Buf), 4, ADR(Rsp_Buf), 4);
```

To read the input word is used the function code 4, followed by 4 bytes of data (two for the start address of the object and two for the amount of words to read). The expected response consists of 3 bytes of data. The first is a input bytes count (in this case 2 because is required 1 word) while the remaining 2 bytes are for the input word requested.
To write the output word is used the function code 6, followed by 4 bytes (two for the start address of the object and two for the value to be written). The expected response is composed of 4 bytes the same as those sent.

Related Topics

MB_RTU_Init

MB_RTU_Deinit

1.16.10 MB_RTU_Wr_Coils

FUNCTION MB_RTU_Wr_Coils : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS RTU function code 15: Write Multiple Coils.

Return Value

0 : success
144 : Quantity parameter not valid
146 : replay with address not valid
148 : replay with quantity not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of first bit to be write

Quantity

Number of bits to be write

Data

Pointer of data buffer for writing

Comments

The function writes the status of one or more bits of memory (Coils) into a specific slave using the MODBUS RTU protocol.
This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the starting bit address to write.
Quantity parameter indicates the number of bits to be write.
Data parameter is the pointer of the buffer containing the values of the bits to be written. The LSB of the first byte of the buffer contains the status to write into the addressed bit. The number of bytes in the buffer to be sent must be equal to: Data_Len = Quantity / 8, where if remainder > 0 then Data_Len = Data_Len + 1.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Rd_Coils

MB_RTU_Wr_Single_Coil

1.16.11 MB_RTU_Wr_Hold_Regs

FUNCTION MB_RTU_Wr_Hold_Regs : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS RTU function code 16: Write Multiple Registers.

Return Value

0 : success
144 : Quantity parameter not valid
146 : replay with address not valid
148 : replay with quantity not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of first word to be write

Quantity

Number of words to be write

Data

Pointer of data buffer for writing

Comments

The function writes the status of one or more words of memory (Holding Registers) into a specific slave using the MODBUS RTU protocol.
This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the starting word address to write.
Quantity parameter indicates the number of words to be write.
Data parameter is the pointer of the buffer containing the values of the words to be written. The number of bytes in the buffer to be sent must be equal to: Data_Len = 2 * Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Rd_Hold_Regs

MB_RTU_Wr_Single_Reg

1.16.12 MB_RTU_Wr_Rd_Hold_Regs

FUNCTION MB_RTU_Wr_Rd_Hold_Regs : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Wr_Address : *WORD*;
  Wr_Quantity : *WORD*;
  Wr_Data : *DWORD*;
  Rd_Address : *WORD*;
  Rd_Quantity : *WORD*;
  Rd_Data : *DWORD*;
END_VAR

Executes the MODBUS RTU function code 23: Write/Read Multiple Registers.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Wr_Address

Address of first word to be write

Wr_Quantity

Number of words to be write

Wr_Data

Pointer of data buffer for writing

Rd_Address

Address of first word to be read

Rd_Quantity

Number of words to be read

Rd_Data

Pointer of data buffer for reading

Comments

The function simultaneously writes and reads the value of one or more words of memory (Holding Registers) into and from a specific slave using the MODBUS RTU protocol. This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Wr_Address parameter is the starting word address to write.
Wr_Quantity parameter indicates the number of words to be write.
Wr_Data parameter is the pointer of the buffer containing the values of the words to be written. The number of bytes in the buffer to be sent must be equal to: Wr_Data_Len = 2 * Wr_Quantity.
Rd_Address parameter is the starting word address to read.
Rd_Quantity parameter indicates the number of words to be read.
Rd_Data parameter is the pointer of the buffer in which receive the values of the words. The amount of bytes placed in the receive buffer is equal to: Rd_Data_Len = 2 * Rd_Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Rd_Hold_Regs

MB_RTU_Wr_Hold_Regs

1.16.13 MB_RTU_Wr_Single_Coil

FUNCTION MB_RTU_Wr_Single_Coil : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Value : *WORD*;
END_VAR

Executes the MODBUS RTU function code 5: Write Single Coil.

Return Value

0 : success
145 : Value parameter not valid
146 : replay with address not valid
149 : replay with value not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of bit to be write

Value

State to write into the bit: 16#0000=OFF, 16#FF00=ON

Comments

The function writes the status of single bit of memory (Coil) into a specific slave using the MODBUS RTU protocol.
This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the bit address to write.
Value parameter indicates the status to be written into the bit. A value of 16#0000 forces the bit in the OFF state while the value 16#FF00 force the bit ON.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Rd_Coils

MB_RTU_Wr_Coils

1.16.14 MB_RTU_Wr_Single_Reg

FUNCTION MB_RTU_Wr_Single_Reg : UINT

VAR_INPUT
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Value : *WORD*;
END_VAR

Executes the MODBUS RTU function code 6: Write Single Register.

Return Value

0 : success
146 : replay with address not valid
149 : replay with value not valid

Input Parameters

Slave_Add

Slave address (valid 1-247)

Address

Address of word to be write

Value

Value to write into the word

Comments

The function writes the value of single word of memory (Holding Register) into a specific slave using the MODBUS RTU protocol.
This is based on MB_RTU_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Slave_Add parameter selects the specific slave on serial bus.
Address parameter is the word address to write.
Value parameter indicates the value to be written into the word.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_RTU_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_RTU_Rd_Hold_Regs

MB_RTU_Wr_Hold_Regs

1.16.15 MB_RTU_Slave

FUNCTION BLOCK MB_RTU_Slave

```
VAR_INPUT
   Start : BOOL;
   Port : PORTS;
   Baudrate : UDINT;
   Parity : MB_RTU_PARITY;
   Slave_Add : BYTE;
   Coils_Ptr : POINTER TO BYTE;
   Coils_Max : UINT;
   Inputs_Ptr : POINTER TO BYTE;
   Inputs_Max : UINT;
   Hold_Regs_Ptr : POINTER TO WORD;
   Hold_Regs_Max : UINT;
   Input_Regs_Ptr : POINTER TO WORD;
   Input_Regs_Max : UINT;
END_VAR

VAR_OUTPUT
   Ready : BOOL;
   Status : MODBUS_STATUS;
END_VAR

VAR
   State : INT;
   Frame_Timeout : UDINT;
   Baud_Low : BOOL;
   Delete_Echo : BOOL;
END_VAR
```

Function block to establish a permanent MODBUS RTU slave service.

Input Variables

Start

Activation signal.

Port

Serial port number. Default to COM2.

Baudrate

Speed of communication (300-1000000 bits/s). Default to 9600.

Parity

Parity select (0/1/2=NO/ODD/EVEN). Default to PARITY_EVEN.

Slave_Add

Slave address (valid 1-247). Default to 1.

Coils_Ptr

Pointer to bit addressable area for Coil elements

Coils_Max

Maximum number of Coil elements

Inputs_Ptr

Pointer to bit addressable area for Discrete Input elements

Inputs_Max

Maximum number of Discrete Input elements

Hold_Regs_Ptr

Pointer to word addressable area for Holding Register elements

Hold_Regs_Max

Maximum number of Holding Register elements

Input_Regs_Ptr

Pointer to word addressable area for Input Register elements

Input_Regs_Max

Maximum number of Input Register elements

Output Variables

Ready

Service ready flag

Status

Current status of the service

Internal Variables

State

Current status of the state graph of the function block

Frame_Timeout

Calculated maximum time (x 1us) for frame receiving

Baud_Low

Selected baudrate is less/equal to 115200

Delete_Echo

Deleting of RS485 driver echo (automatically defined on specific hardware)

Comments

This function block installs a permanent slave service with MODBUS RTU protocol. The function block, once activated with the Start signal, initializes the communication and then enter into a state of service activity. In the active state, the block awaits the arrival of a command frame from MODBUS master. When the frame is received, it is checked and, if the command is correct for the specific slave, the response is transmitted. To deactivate the service the Start signal must be forced FALSE.
Port parameter identifies the serial port (1-3) used for communication.
Coils_Ptr and Coils_Max parameters respectively define the pointer to the first byte of the area containing the bits associated with the Coils block of the MODBUS memory model and the maximum number of addressable bits. The number of bits may not be a multiple of 8, but the first bit must necessarily be aligned with the LSB of the first byte.
Similarly Inputs_Ptr and Inputs_Max parameters work relatively to the area containing the bits associated with Discrete inputs block of the MODBUS memory model.
Hold_Regs_Ptr and Hold_Regs_Max parameters respectively define the pointer to the first word of the area containing the words associated with the Holding Registers block of the MODBUS memory model and the maximum number of addressable words.
Similarly Input_Regs_Ptr and Input_Regs_Max parameters work relatively to the area containing the words associated with the Input Registers block of the MODBUS memory model.

NOTE: all the elements of each block of the MODBUS model are contiguous and addressed from 0 coinciding with the initial position of the memory area pointed by the supplied parameters.

Related Topics

MODBUS_STATUS

1.16.16 MB_TCP_Connect

FUNCTION MB_TCP_Connect : UINT

```
VAR_INPUT
   IP_Address : STRING;
   Port : UINT;
   Keepalive_En : BOOL;
   Keepalive_Time : INT;
   Keepalive_Intv : INT;
   Keepalive_Retry : INT;
   Rsp_Timeout : UINT;
   Socket : POINTER TO INT;
END_VAR
```

Initializes and connect MODBUS TCP/IP client to a specific server.

Return Value

0 : success
128 : initialization parameter not valid
129 : inizialization error
137 : reached the maximum number of instances (32)
>200 : error code according to the coding adopted by the TCP/IP stack

Input Parameters

IP_Address

Network address of the server to connect to

Port

Communication TCP/IP port of the server

Keepalive_En

Activation of Keepalive feature

Keepalive_Time

Idle time (s) before Probes sending (10-32767)

Keepalive_Intv

Time (s) interval between Probes (1-600)

Keepalive_Retry

Maximun number of Probes before connection close (0-32767)

Rsp_Timeout

Maximum wait (ms) for server response reception (for example 1000)

Socket

Returned socket descriptor to use as connection handle

Comments

This function initializes a MODBUS TCP/IP client connection with a server at a specific IP address. This function must be called before start a communication session.
Up to 32 instances of simultaneous connection are possible with different socket handles.
IP_Address parameter identifies the network address of the server to which to connect.
The Port parameter identifies the TCP port used for connection. Normally the 502 port is used for MODBUS TCP/IP.
The Keepalive parameters set the feature of sending dummy messages (Probes) in order to check whether the connection with the counterpart (server peer) is still active. If Keepalive_En is enabled and the connection is inactive for a time equal to Keepalive_Time,

begins sending periodic TCP packets with no data (Probes) with a repetition time equal to Keepalive_Intv. After Keepalive_Retry maximum attempts, if no answer is received from the connected device, the connection is automatically closed.
The Rsp_Timeout parameter sets the maximum waiting time in ms for the response from the server.
The function returns 0 if executed correctly. In this case the value returned on the Socket variable must be saved for use it in all other client functions as a handle of the specific connection. In case of error the function returns a value > 0 that encodes the error.

Related Topics

MB_TCP_Disconnect

1.16.17 MB_TCP_Disconnect

FUNCTION MB_TCP_Disconnect : UINT

```
VAR_INPUT
   Socket : INT;
END_VAR
```

Disconnect MODBUS TCP/IP client from a specific server.

Return Value

0 : success
>200 : error code according to the coding adopted by the TCP/IP stack

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Comments

This function disconnects the MODBUS TCP/IP client from the server. This function must be called at the end of the MODBUS session.
If Socket parameter value is -1, all previously opened connection instances will be closed.
If executed correctly returns 0 otherwise it returns the corresponding TCP/IP error code.

Related Topics

MB_TCP_Connect

1.16.18 MB_TCP_Rd_Coils

FUNCTION MB_TCP_Rd_Coils : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS TCP/IP function code 1: Read Coils.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of first bit to be read

Quantity

Number of bits to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more bits of memory (Coils) from a specific server/slave using the MODBUS TCP/IP protocol. This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the starting bit address to read.
Quantity parameter indicates the number of bits to be read.
Data parameter is the pointer of the buffer in which receive the values of the bits. The LSB of the first byte of the buffer contains the addressed bit. The amount of bytes placed in the receive buffer is equal to: Data_Len = Quantity / 8, where if remainder > 0 then Data_Len = Data_Len + 1.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Wr_Coils

MB_TCP_Wr_Single_Coil

1.16.19 MB_TCP_Rd_Hold_Regs

FUNCTION MB_TCP_Rd_Hold_Regs : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS TCP/IP function code 3: Read Holding Registers.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of first word to be read

Quantity

Number of words to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more words of memory (Holding Registers) from a specific server/slave using the MODBUS TCP/IP protocol.
This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the starting word address to read.
Quantity parameter indicates the number of words to be read.
Data parameter is the pointer of the buffer in which receive the values of the words. The amount of bytes placed in the receive buffer is equal to: Data_Len = 2 * Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Wr_Hold_Regs

MB_TCP_Wr_Single_Reg

1.16.20 MB_TCP_Rd_Input_Regs

FUNCTION MB_TCP_Rd_Input_Regs : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS TCP/IP function code 4: Read Input Registers.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of first word to be read

Quantity

Number of words to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more words of input (Input Registers) from a specific server/slave using the MODBUS TCP/IP protocol.
This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the starting word address to read.
Quantity parameter indicates the number of words to be read.
Data parameter is the pointer of the buffer in which receive the values of the words. The amount of bytes placed in the receive buffer is equal to: Data_Len = 2 * Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Rd_Inputs

1.16.21 MB_TCP_Rd_Inputs

FUNCTION MB_TCP_Rd_Inputs : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS TCP/IP function code 2: Read Discrete Inputs.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of first bit to be read

Quantity

Number of bits to be read

Data

Pointer of data buffer for reading

Comments

The function reads the status of one or more bits of input (Discrete Inputs) from a specific server/slave using the MODBUS TCP/IP protocol.
This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the starting bit address to read.
Quantity parameter indicates the number of bits to be read.
Data parameter is the pointer of the buffer in which receive the values of the bits. The LSB of the first byte of the buffer contains the addressed bit. The amount of bytes placed in the receive buffer is equal to: Data_Len = Quantity / 8, where if remainder > 0 then Data_Len = Data_Len + 1.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Rd_Input_Regs

1.16.22 MB_TCP_Req_Rsp

FUNCTION MB_TCP_Req_Rsp : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Fun_Code : *BYTE*;
  Req_Data : *DWORD*;
  Req_Len : *WORD*;
  Rsp_Data : *DWORD*;
  Rsp_Len : *WORD*;
END_VAR

Executes a complete MODBUS TCP/IP request tx and response rx with a connected server.

Return Value

0 : success
1 : slave has received a not valid function code
2 : slave has received a not valid object address
3 : slave has received a not valid object data
4 : non-recoverable error during execution of the slave function
5 : response to prevent a master receive timeout while slave performs a long operation
6 : slave is busy in the execution of required function
130 : timeout during master reception
133 : bad slave address in replay
134 : bad function code in replay
136 : bad number of bytes received
138 : bad MBAP header in replay
>200 : error code according to the coding adopted by the TCP/IP stack

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Fun_Code

MODBUS function code

Req_Data

Pointer to request data buffer

Req_Len

Number of request data bytes

Rsp_Data

Pointer to response data buffer

Rsp_Len

Number of response data bytes aspected

Comments

This function execute a complete MODBUS TCP/IP communication sequence with a server on the network.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter selects the specific ID of a slave connected on a subnet over serial bus. This is used in the case of complex networks realized with a gateway that convert between Modbus TCP/IP and RTU to address a specific slave within the subnet. The

values 0-247 are normally assigned to a slave while the value 255 is used for the direct communication with the gateway or in the absence of subnets.
Fun_Code parameter is the code of the function to execute.
Req_Data parameter is the pointer to the buffer with the data to be transmitted for the specific function code.
Req_Len parameter indicates the number of bytes in the Req_Data buffer to be transmitted to the slave.
Rsp_Data parameter is the pointer to the buffer in which receive the data of response to the function from the slave.
The Rsp_Len parameter indicates the number of data bytes expected in the Rsp_Data receive buffer as response to the specific function code.
The function returns 0 if performed without error, otherwise it returns the error code.
To use the MODBUS communication function refer to the technical documentation "MODBUS Application Protocol Specification" downloadable at http://www.modbus-ida.org./tech.php.

In the following example one input word and one output word are refreshed for the server 192.168.1.102:

```
PROGRAM MB_TCP
VAR
    MB_TCP_Connected: BOOL := FALSE;
    MB_TCP_Socket: INT;
    Req_Buf: ARRAY[0..10] OF BYTE;
    Rsp_Buf: ARRAY[0..10] OF BYTE;
    Input_Reg: WORD;
    Output_Reg: WORD;
END_VAR

(* Initialize MODBUS connection *)
IF NOT MB_TCP_Connected THEN
    IF MB_TCP_Connect('192.168.1.102', 502, TRUE, 7200, 75, 8, 1000,
ADR(MB_TCP_Socket)) = 0 THEN
        MB_TCP_Connected := TRUE;
    END_IF
END_IF

(* Read one input word at address 0 from slave 255 (no RTU subnet) *)
Req_Buf[0] := 0;
Req_Buf[1] := 0;
Req_Buf[2] := 0;
Req_Buf[3] := 1;
IF MB_TCP_Connected THEN
    IF MB_TCP_Req_Rsp(MB_TCP_Socket, 255, 4, ADR(Req_Buf), 4, ADR(Rsp_Buf), 3) = 0
THEN
        Input_Reg := 256 * Rsp_Buf[1] + Rsp_Buf[2];
    END_IF
END_IF

(* Write one output word at address 1 to Slave 255 (no RTU subnet) *)
Req_Buf[0] := 0;
Req_Buf[1] := 1;
Req_Buf[2] := Output_Reg / 256;
Req_Buf[3] := Output_Reg MOD 256;
IF MB_TCP_Connected THEN
    MB_TCP_Req_Rsp(MB_TCP_Socket, 255, 6, ADR(Req_Buf), 4, ADR(Rsp_Buf), 4);
END_IF
```

To read the input word is used the function code 4, followed by 4 bytes of data (two for the start address of the object and two for the amount of words to read). The expected response consists of 3 bytes of data. The first is a input bytes count (in this case 2 because is required 1 word) while the remaining 2 bytes are for the input word requested.
To write the output word is used the function code 6, followed by 4 bytes (two for the start address of the object and two for the value to be written). The expected response is composed of 4 bytes the same as those sent.

Related Topics

MB_TCP_Connect

MB_TCP_Disconnect

1.16.23 MB_TCP_Wr_Coils

FUNCTION MB_TCP_Wr_Coils : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS TCP/IP function code 15: Write Multiple Coils.

Return Value

0 : success
144 : Quantity parameter not valid
146 : replay with address not valid
148 : replay with quantity not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of first bit to be write

Quantity

Number of bits to be write

Data

Pointer of data buffer for writing

Comments

The function writes the status of one or more bits of memory (Coils) into a specific server/slave using the MODBUS TCP/IP protocol. This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the starting bit address to write.
Quantity parameter indicates the number of bits to be write.
Data parameter is the pointer of the buffer containing the values of the bits to be written. The LSB of the first byte of the buffer contains the status to write into the addressed bit. The number of bytes in the buffer to be sent must be equal to: Data_Len = Quantity / 8, where if remainder > 0 then Data_Len = Data_Len + 1.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Rd_Coils

MB_TCP_Wr_Single_Coil

1.16.24 MB_TCP_Wr_Hold_Regs

FUNCTION MB_TCP_Wr_Hold_Regs : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Quantity : *WORD*;
  Data : *DWORD*;
END_VAR

Executes the MODBUS TCP/IP function code 16: Write Multiple Registers.

Return Value

0 : success
144 : Quantity parameter not valid
146 : replay with address not valid
148 : replay with quantity not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of first word to be write

Quantity

Number of words to be write

Data

Pointer of data buffer for writing

Comments

The function writes the status of one or more words of memory (Holding Registers) into a specific server/slave using the MODBUS TCP/IP protocol.
This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the starting word address to write.
Quantity parameter indicates the number of words to be write.
Data parameter is the pointer of the buffer containing the values of the words to be written. The number of bytes in the buffer to be sent must be equal to: Data_Len = 2 * Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Rd_Hold_Regs

MB_TCP_Wr_Single_Reg

1.16.25 MB_TCP_Wr_Rd_Hold_Regs

FUNCTION MB_TCP_Wr_Rd_Hold_Regs : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Wr_Address : *WORD*;
  Wr_Quantity : *WORD*;
  Wr_Data : *DWORD*;
  Rd_Address : *WORD*;
  Rd_Quantity : *WORD*;
  Rd_Data : *DWORD*;
END_VAR

Executes the MODBUS TCP/IP function code 23: Write/Read Multiple Registers.

Return Value

0 : success
144 : Quantity parameter not valid
147 : replay with byte counter value not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Wr_Address

Address of first word to be write

Wr_Quantity

Number of words to be write

Wr_Data

Pointer of data buffer for writing

Rd_Address

Address of first word to be read

Rd_Quantity

Number of words to be read

Rd_Data

Pointer of data buffer for reading

Comments

The function simultaneously writes and reads the value of one or more words of memory (Holding Registers) into and from a specific server/slave using the MODBUS TCP/IP protocol. This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Wr_Address parameter is the starting word address to write.
Wr_Quantity parameter indicates the number of words to be write.
Wr_Data parameter is the pointer of the buffer containing the values of the words to be written. The number of bytes in the buffer to be sent must be equal to: Wr_Data_Len = 2 * Wr_Quantity.

Rd_Address parameter is the starting word address to read.
Rd_Quantity parameter indicates the number of words to be read.
Rd_Data parameter is the pointer of the buffer in which receive the values of the words. The amount of bytes placed in the receive buffer is equal to: Rd_Data_Len = 2 * Rd_Quantity.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Rd_Hold_Regs

MB_TCP_Wr_Hold_Regs

1.16.26 MB_TCP_Wr_Single_Coil

FUNCTION MB_TCP_Wr_Single_Coil : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Value : *WORD*;
END_VAR

Executes the MODBUS TCP/IP function code 5: Write Single Coil.

Return Value

0 : success
145 : Value parameter not valid
146 : replay with address not valid
149 : replay with value not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of bit to be write

Value

State to write into the bit: 16#0000=OFF, 16#FF00=ON

Comments

The function writes the status of single bit of memory (Coil) into a specific server/slave using the MODBUS TCP/IP protocol.
This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the bit address to write.
Value parameter indicates the status to be written into the bit. A value of 16#0000 forces the bit in the OFF state while the value 16#FF00 force the bit ON.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Rd_Coils

MB_TCP_Wr_Coils

1.16.27 MB_TCP_Wr_Single_Reg

FUNCTION MB_TCP_Wr_Single_Reg : UINT

VAR_INPUT
  Socket : *INT*;
  Slave_Add : *BYTE*;
  Address : *WORD*;
  Value : *WORD*;
END_VAR

Executes the MODBUS TCP/IP function code 6: Write Single Register.

Return Value

0 : success
146 : replay with address not valid
149 : replay with value not valid

Input Parameters

Socket

Socket descriptor (handle) used by the connection

Slave_Add

Slave address (valid 0-255)

Address

Address of word to be write

Value

Value to write into the word

Comments

The function writes the value of single word of memory (Holding Register) into a specific server/slave using the MODBUS TCP/IP protocol.
This is based on MB_TCP_Req_Rsp generic communication function for which operates as an interface with the specific command code.
Socket parameter is the handle of the specific connection on which execute the request.
Slave_Add parameter select the specific slave on serial bus if the server is a gateway to a RTU subnet.
Address parameter is the word address to write.
Value parameter indicates the value to be written into the word.
The function returns 0 if it executes without error, otherwise it returns the error code. The error codes are those of the MB_TCP_Req_Rsp function to which are added those of the specific command.

Related Topics

MB_TCP_Rd_Hold_Regs

MB_TCP_Wr_Hold_Regs

1.16.28 MB_TCP_Server

FUNCTION BLOCK MB_TCP_Server

```
VAR_INPUT
   Start : BOOL;
   Port : UINT;
   Keepalive_En : BOOL;
   Keepalive_Time : INT;
   Keepalive_Intv : INT;
   Keepalive_Retry : INT;
   Coils_Ptr : POINTER TO BYTE;
   Coils_Max : UINT;
   Inputs_Ptr : POINTER TO BYTE;
   Inputs_Max : UINT;
   Hold_Regs_Ptr : POINTER TO WORD;
   Hold_Regs_Max : UINT;
   Input_Regs_Ptr : POINTER TO WORD;
   Input_Regs_Max : UINT;
END_VAR

VAR_OUTPUT
   Ready : BOOL;
   Status : MODBUS_STATUS;
END_VAR

VAR
   State : INT;
   Listen_Socket : INT;
   Accepted_Socket : ARRAY[0..7] OF INT;
   Highest_Socket : INT;
   Socket_Flags : ARRAY[0..15] OF WORD;
END_VAR
```

Function block to establish a permanent MODBUS TCP/IP server function.

Input Variables

Start

Activation signal.

Port

TCP/IP port number. Default to 502.

Keepalive_En

Activation of Keepalive feature. Default to TRUE.

Keepalive_Time

Idle time (s) before Probes sending. Range 10-32767, default to 7200.

Keepalive_Intv

Time (s) interval between Probes. Range 1-600, default to 75.

Keepalive_Retry

Maximun number of Probes before connection close. Range 0-32767, default to 8.

Coils_Ptr

Pointer to bit addressable area for Coil elements

Coils_Max

Maximum number of Coil elements

Inputs_Ptr

Pointer to bit addressable area for Discrete Input elements

Inputs_Max

Maximum number of Discrete Input elements

Hold_Regs_Ptr

Pointer to word addressable area for Holding Register elements

Hold_Regs_Max

Maximum number of Holding Register elements

Input_Regs_Ptr

Pointer to word addressable area for Input Register elements

Input_Regs_Max

Maximum number of Input Register elements

Output Variables

Ready

Service ready flag

Status

Current status of the service

Internal Variables

State

Current status of the state graph of the function block

Listen_Socket

Descriptor number of listen socket

Accepted_Socket

List of socket descriptor number of accepted connections

Highest_Socket

Maximum descriptor number of accepted connections

Socket_Flags

List of socket flags of accepted connections

Comments

This function block installs a permanent server service with MODBUS TCP/IP protocol. The function block, once activated with the Start signal, initializes the connection and then enter into a state of service activity. In the active state, the block awaits the arrival of a command frame from MODBUS client. When the frame is received, it is checked and, if the command is correct for the specific server, the response is transmitted. To deactivate the service the Start signal must be forced FALSE.
Port parameter identifies the TCP port used for connection. Normally the 502 port is used for MODBUS TCP/IP.
A server can accept up to a maximum of 8 concurrent connections.
The Keepalive parameters set the feature of sending dummy messages (Probes) in order to check whether the connection with the counterpart (client peer) is still active. If Keepalive_En is enabled and the connection is inactive for a time equal to Keepalive_Time, begins sending periodic TCP packets with no data (Probes) with a repetition time equal to Keepalive_Intv. After Keepalive_Retry maximum attempts, if no answer is received from the connected device, the connection is automatically closed.
Coils_Ptr and Coils_Max parameters respectively define the pointer to the first byte of the area containing the bits associated with the

Coils block of the MODBUS memory model and the maximum number of addressable bits. The number of bits may not be a multiple of 8, but the first bit must necessarily be aligned with the LSB of the first byte.
Similarly Inputs_Ptr and Inputs_Max parameters work relatively to the area containing the bits associated with Discrete inputs block of the MODBUS memory model.
Hold_Regs_Ptr and Hold_Regs_Max parameters respectively define the pointer to the first word of the area containing the words associated with the Holding Registers block of the MODBUS memory model and the maximum number of addressable words.
Similarly Input_Regs_Ptr and Input_Regs_Max parameters work relatively to the area containing the words associated with the Input Registers block of the MODBUS memory model.

NOTE: all the elements of each block of the MODBUS model are contiguous and addressed from 0 coinciding with the initial position of the memory area pointed by the supplied parameters.

Related Topics

MODBUS_STATUS

1.16.29 MB_Swap_Dword

FUNCTION MB_Swap_Dword : DWORD

```
VAR_INPUT
   Value : DWORD;
END_VAR
```

Big-endian/Little-endian conversion for the bytes of a DWORD.

Return Value

Double word value with swapped bytes

Input Parameters

Value

Value of the double word with the bytes to be swapped

Comments

The function performs the conversion from the Big-endian to the Little-endian notation and vice versa for a variable of type DWORD. The double word returned by the function contains the same bytes of the double word passed as a parameter but in reverse order. The PLC processor uses the Little-endian notation while the double words contained in the MODBUS protocol data packets are defined with Big-endian notation. The implementation of custom MODBUS commands with the MB_RTU_Req_Rsp function may require the use of conversion between the two formats. The insertion and estraction of a double word variable, related to a data buffer of type ARRAY OF BYTE, require such conversion function.

Related Topics

MB_Swap_Dword2

MB_Swap_Word

1.16.30 MB_Swap_Dword2

FUNCTION MB_Swap_Dword2 : DWORD

```
VAR_INPUT
   Value : DWORD;
END_VAR
```

Big-endian/Little-endian conversion for the words of a DWORD.

Return Value

Double word value with swapped words

Input Parameters

Value

Value of the double word with the words to be swapped

Comments

The function performs the conversion from the Big-endian to the Little-endian notation and vice versa for a variable of type DWORD. The double word returned by the function contains the same words of the double word passed as a parameter but in reverse order. The PLC processor uses the Little-endian notation while the double words contained in the MODBUS protocol data packets are defined with Big-endian notation. The implementation of custom MODBUS commands with the MB_RTU_Req_Rsp function may require the use of conversion between the two formats. The insertion and estraction of a double word variable, related to a data buffer of type ARRAY OF BYTE, require such conversion function.

Related Topics

MB_Swap_Dword

MB_Swap_Word

1.16.31 MB_Swap_Word

FUNCTION MB_Swap_Word : WORD

```
VAR_INPUT
   Value : WORD;
END_VAR
```

Big-endian/Little-endian conversion for the bytes of a WORD.

Return Value

Word value with swapped bytes

Input Parameters

Value

Value of the word with the bytes to be swapped

Comments

The function performs the conversion from the Big-endian to the Little-endian notation and vice versa for a variable of type WORD. The word returned by the function contains the same bytes of the word passed as a parameter but in reverse order.
The PLC processor uses the Little-endian notation while the words contained in the MODBUS protocol data packets are defined with Big-endian notation. The implementation of custom MODBUS commands with the MB_RTU_Req_Rsp function may require the use of conversion between the two formats. The insertion and estraction of a word variable, related to a data buffer of type ARRAY OF BYTE, require such conversion function.
In the case of the standard MODBUS commands, already implemented with specific functions, the word data conversions are handled by the functions itself.

Related Topics

MB_Swap_Dword

MB_Swap_Dword2

2. PLC Browser Commands

The PLC Browser (i.e. command line interpreter) is extended by some commands specific to the target. These commands include checking the versions of the operating system and run-time kernel and setting and checking the IP configuration.
To execute a command, you must be logged into the controller. Then change to the PLC Browser by clicking on the "Resources" tab of the project tree. Then double-click on "PLC-Browser", type the command into the first line of the PLC browser window and press the Enter key. The controller's response to the command will be displayed in the lower area of the PLC-Browser window.

Here is a list of all extended commands available:

DHCP
DNSNAME
FTPPASSWORD
FTPUSER
GATEWAY
GETDATETIME
IP
IPCFG
IPETH
NETMASK
PING
REBOOT
SETDATETIME
TCPIPMEM
VER
WEBPASSWORD
WEBUSER

2.1 DHCP

The DHCP command enables or disables the use of the Dynamic Host Configuration Protocol for the internal ethernet interface to obtain an IP configuration.
When DHCP is enabled, it is not necessary to set the IP configuration manually using the commands IP, NETMASK and GATEWAY.
After setting the IP configuration or turning DHCP on or off, the ethernet interface must be reinitialized using the IPETH command.

Command Syntax: `dhcp x` where x is either 0 (DHCP off) or 1 (DHCP on)

Example: `dhcp 1`

Related Topics

GATEWAY command

IP command

IPETH command

NETMASK command

2.2 DNSNAME

Change the current value of [DNS] NAME_SERVER1/2 entry into CHIP.INI file.
DNS server is a service to resolve the IP address value from a registered domain name.
NOTE: two name setting is available (1 and 2).

Command Syntax: `dnsname n xxx.xxx.xxx.xxx`

Example: `dnsname 1 8.8.8.8`

2.3 FTPPASSWORD

Change the current value of [FTP] PASSWORD0/1 entry into CHIP.INI file.
User name and password registration protects access to FTP connection.
NOTE: two user setting is available (0 and 1).

Command Syntax: `ftppassword n password`

Example: `ftppassword 0 ftp_password0`

Related Topics

FTPUSER command

2.4 FTPUSER

Change the current value of [FTP] USER0/1 entry into CHIP.INI file.
User name and password registration protects access to FTP connection.
NOTE: two user setting is available (0 and 1).

Command Syntax: `ftpuser n user_name`

Example: `ftpuser 0 ftp_user0`

Related Topics

FTPPASSWORD command

2.5 GATEWAY

The GATEWAY command sets the address of the standard gateway of the ethernet interface.
After setting the IP configuration, the ethernet interface must be reinitialized using the IPETH command.

Command Syntax: `gateway xxx.xxx.xxx.xxx`

Example: `gateway 192.168.1.1`

Related Topics

IP command

IPETH command

NETMASK command

2.6 GETDATETIME

The GETDATETIME command displays the current Date and Time of controller.

Command Syntax: `getdatetime`

Related Topics

SETDATETIME command

2.7 IP

The IP command sets the IP address of the controller's ethernet interface. After setting the IP configuration, the ethernet interface must be reinitialized using the IPETH command.

Command Syntax: `ip xxx.xxx.xxx.xxx`

Example: `ip 192.168.1.101`

Related Topics

GATEWAY command

IPETH command

NETMASK command

2.8 IPCFG

The IPCFG command displays the current IP configuration of the controller. For each network interface, the device name, IP address, subnet mask and related information are displayed.

Command Syntax: `ipcfg`

2.9 IPETH

The IPETH command reconfigures the ethernet interface after the IP configuration has been changed using the IP, NETMASK, GATEWAY, or DHCP command. If DHCP is enabled, IPETH also obtains a new IP configuration from the DHCP Server.

Command Syntax: `ipeth`

Related Topics

DHCP command

GATEWAY command

IP command

NETMASK command

2.10 NETMASK

The NETMASK command sets the subnet mask of the ethernet interface.
After setting the IP configuration, the ethernet interface must be reinitialized using the IPETH command.

Command Syntax: `netmask xxx.xxx.xxx.xxx`

Example: `netmask 255.255.255.0`

Related Topics

GATEWAY command

IP command

IPETH command

2.11 PING

The PING command allows to check if a particular other computer can be reached via the network.
After starting PING, four packets of data are sent to the other computer, which will then send a reply back to the controller. The result of the PING (number of packets successfully received) will then be displayed in the PLC Browser. Note that while waiting for the reply, a series of numbers is output in the PLC Browser to indicate that the command is still in process.

Command Syntax: `ping xxx.xxx.xxx.xxx`

Example: `ping 192.168.1.2`

2.12 REBOOT

This command reboot the CPU and restart the execution of all services and programs.
NOTE: after this comand the communication with PLC is lost for some seconds.

Command Syntax: `reboot`

2.13 SETDATETIME

The SETDATETIME command writes the updated Date and Time into controller.
The format of Date and Time is the same of DT data type.

Command Syntax: `setdatetime yyyy-mm-dd-hh:mm:ss`

Example: `setdatetime 2010-04-29-16:35:40`

Related Topics

GETDATETIME command

2.14 TCPIPMEM

The TCPIPMEM command displays TCP/IP memory usage. This command shows the maximum reserved memory for the TCP/IP stack and the current TCP/IP stack memory used.

Command Syntax: `tcpipmem`

2.15 VER

The VER command displays the version numbers of the controller's operating system and of the controller's run-time kernel.

Command Syntax: `ver`

2.16 WEBPASSWORD

Change the current value of [WEB] SEC_PASSWORD0/1 entry into CHIP.INI file.
User name and password registration protects access to WEB server connection.
NOTE: two user setting is available (0 and 1).

Command Syntax: `webpassword n password`

Example: `webpassword 0 web_password0`

Related Topics

WEBUSER command

2.17 WEBUSER

Change the current value of [WEB] SEC_USER0/1 entry into CHIP.INI file.
User name and password registration protects access to WEB server connection.
NOTE: two user setting is available (0 and 1).

Command Syntax: `webuser n user_name`

Example: `webuser 0 web_user0`

Related Topics

WEBPASSWORD command

3. Error handling

The CoDeSys "error handling" concept.

CoDeSys uses events to handle runtime errors. A runtime error is an exception that occurs during the execution of an IEC project that disrupts the normal flow of instructions. The occurrence of the exception makes the execution of the following intruction arguable, so the IEC project will be stopped automaticly on the end of the current cycle. By registering an event handler it is possible to execute any IEC code. Bevor you restart your program you have to reset it. Only when the exception EVENT_EXCPT_ETHLINKLOST occurs you can restart your program without a reset.
All runtime errors are listed in the CoDeSys task configuration dialog.

The following exceptions (runtime errors) could be generated by the CoDeSys RTS:

1. `EVENT_EXCPT_DIVIDEBYZERO:` Division by zero (on integer datatypes)
2. `EVENT_EXCPT_FPU_DIVIDEBYZERO:` Division by zero (on floating point datatypes)
3. `EVENT_EXCPT_FPU_INVALID_OPERATION:` Floatingpoint operations with invalid input value (e.g. radical from -1)
4. `EVENT_EXCPT_NETWORKERROR:` Fatal network/TCP-IP error (mostly an memory problem)
5. `EVENT_EXCPT_HARDWAREERROR:` Fatal hardware error (e.g. flash defect, ethernet controller defect)
6. `EVENT_EXCPT_OUTOFMEMORY:` Not enough memory available
7. `EVENT_EXCPT_ILLEGAL_INSTRUCTION:` Unknown processor instruction
8. `EVENT_EXCPT_KERNELERROR:` Internal kernel error
9. `EVENT_EXCPT_ETHLINKLOST:` Ethernet link lost